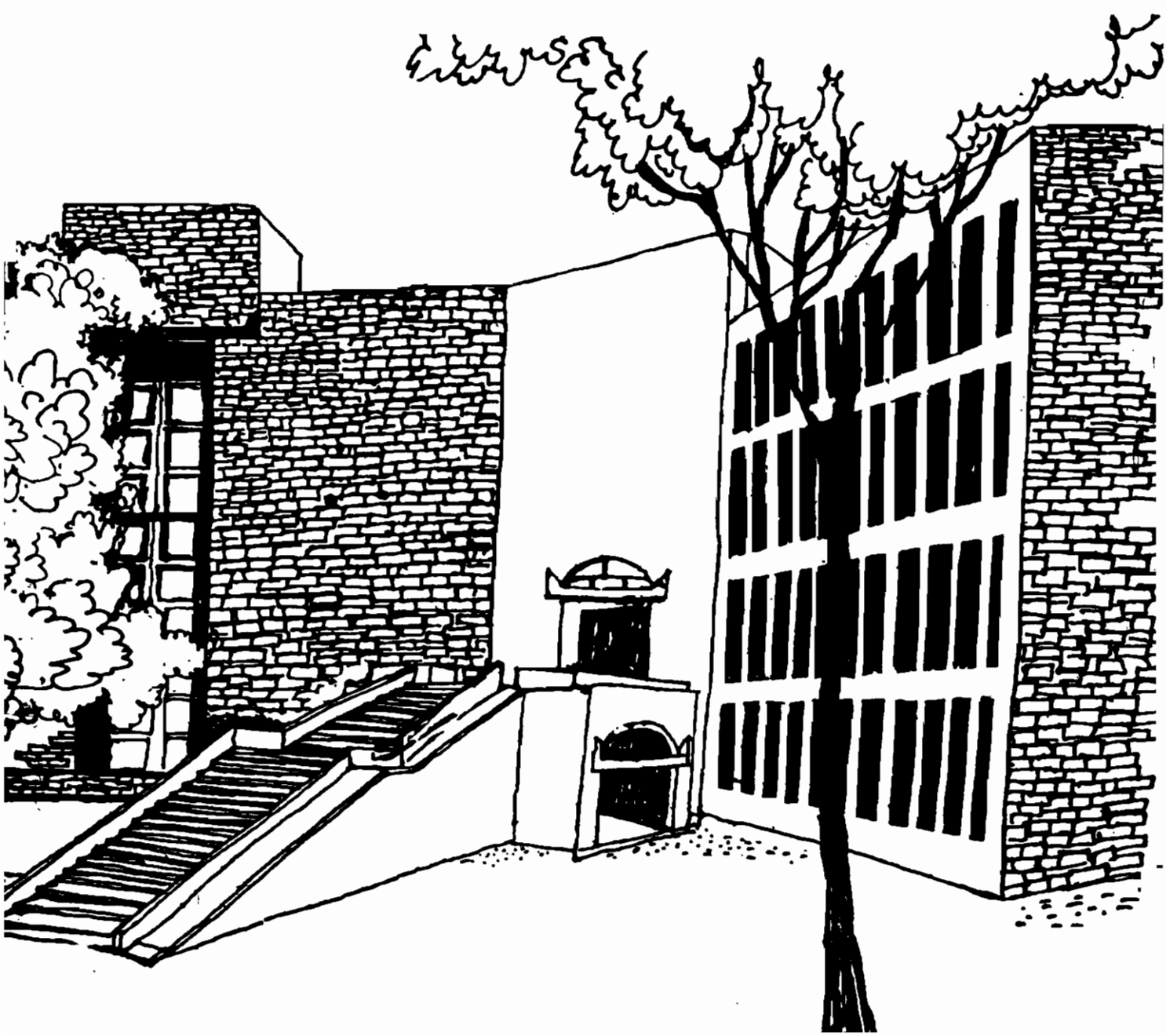
विद्याविनियोगाद्विकासः

IIM

AHMEDABAD

Working Paper



DATA CORRECTING: A METHODOLOGY FOR OBTAINING NEAR-OPTIMAL SOLUTIONS

Diptesh Ghosh
Boris Goldengorin
Gerard Sierksma

W.P.No.2002-06-01
June 2002 1703

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380 015
INDIA

DATA CORRECTING: A METHODOLOGY FOR OBTAINING NEAR-OPTIMAL SOLUTIONS

DIPTESH GHOSH, BORIS GOLDENGORIN, AND GERARD SIERKSMA

ABSTRACT. In this paper we concern ourselves with the problem of finding near-optimal solutions to functions that are not amenable to solution by analytic means. This means that given a function and a parameter α, we provide a technique that yield a solution whose function value is no more than α away from the optimal value of the function. We show how this technique can be applied to combinatorial optimization problems and present our computational experience with this algorithm on benchmark asymmetric traveling salesperson problems.

Keywords: data correcting, polynomially solvable special cases, combinatorial optimization, asymmetric traveling salesperson problems.

1. INTRODUCTION

Polynomially solvable special cases have long been studied in the literature on combinatorial optimization problems (see, for example, [6]). Apart from being mathematical curiosities, they often provide important insights for serious problem-solving. In fact, the concluding paragraph of [6] states the following, regarding polynomially solvable special cases for the traveling salesperson problem.

> " ... We believe, however, that in the long run the greatest importance of these special cases will be for approximation algorithms. Much remains to be done in this area."

This paper is a step in the direction of incorporating polynomially solvable special cases into approximation algorithms. We propose a *data correcting algorithm* — an approximation algorithm that makes use of polynomially solvable special cases to arrive at high-quality solutions. The basic insight that leads to this algorithm is the fact that it is often easy to compute a bound on the difference between the cost of an optimal solution to a problem instance, and of the optimal solution for any other instance even though it may be hard to compute an optimal solution for the latter instance. The results obtained with this algorithm are very promising (see the results for the traveling salesperson problem in Section 4 in this paper, and for the quadratic cost partitioning problem in [7]). Although we have already carried out some more experimentation with this algorithm (see, for example, [8]), a thorough introduction on the data correcting algorithm has not appeared in the literature. This paper fills that gap.

The approximation in the data correcting algorithm is in terms of an *accuracy parameter*, which is an upper bound on the difference between the objective of an optimal solution to the instance and that of a solution returned by the data correcting

algorithm. Note that this is not expressed as a fraction of the optimal objective value for this instance. In this respect, the algorithm is different from common ε-optimal algorithms, in which ε is defined as a fraction of the optimal objective value.

Even though we use the data correcting algorithm mainly for solving NP-hard combinatorial optimization problems, it can be used for functions defined on a continuous domain too. We will in fact, motivate the algorithm in the next section using a function defined on a continuous domain, and having a finite range. We then show in Section 3, how this approach can be adapted for combinatorial optimization problems. We illustrate this adaptation with the help of an example in this section. In Section 4 we present our computational experience with this method on asymmetric traveling salesperson problem instances from the TSPLIB [11]. We conclude the paper with a summary and discussions for future research in Section 5.

2. Data Correcting for Real-Valued Functions

Consider a real-valued function $f : \mathcal{D} \to \Re$, where $\mathcal{D}$ is the domain on which the function is defined. We assume that f is not analytically tractable, and concern ourselves with the problem of finding α-minimal solutions to the function f over D, i.e. the problem of finding a member of $\{x | x \in \mathcal{D}, f(x) \leq f(x^*) + \alpha\}$, where $x^* = \arg\min_{\mathcal{D}}\{f(x)\}$, and α is a pre-defined *accuracy* parameter. The discussion here is for a minimization problem, the maximization problem can be dealt with in a similar manner.

Let us assume a partition $\{\mathcal{D}_1, \ldots, \mathcal{D}_p\}$ of the domain $\mathcal{D}$. Let us further assume that for each of the sub-domains $\mathcal{D}_i$ of $\mathcal{D}$, we are able to find functions $g_i : \mathcal{D}_i \to \Re$, which are easy to minimize over $\mathcal{D}_i$, and such that

$$|f(x) - g_i(x)| \leq \frac{\alpha}{2} \quad \forall x \in \mathcal{D}_i. \tag{1}$$

We call such easily minimizable functions *regular*.

Theorem 1 demonstrates an important relationship between the regular functions g_i and the original function f. It states that the function value of f at the best among the minima of all the g_i's over their respective domains is close to the minimum function value of f over the domain.

Theorem 1. *Let* $x_i^\alpha \in \arg\min_{x \in \mathcal{D}_i}\{g_i(x)\}$, $x^\alpha \in \arg\min_i\{f(x_i^\alpha)\}$, *and let* $x^* \in \arg\min_{x \in \mathcal{D}}\{f(x)\}$. *Then*

$$f(x^\alpha) \leq f(x^*) + \alpha.$$

Proof. Let $x_i^* \in \arg\min_{x \in \mathcal{D}_i}\{f(x)\}$. Then for $i = 1, \ldots, p$, $f(x_i^\alpha) - \frac{\alpha}{2} \leq g_i(x_i^\alpha) \leq g_i(x_i^*) \leq f(x_i^*) + \frac{\alpha}{2}$, i.e. $f(x_i^\alpha) \leq f(x_i^*) + \alpha$. Thus $\min_i\{f(x_i^\alpha)\} \leq \min_i\{f(x_i^*)\} + \alpha$, which proves the result. □

Notice that x^* and x^α do not need to be in the same sub-domain of $\mathcal{D}$.

Theorem 1 forms the basis of the data correcting algorithm to find an approximate minimum of a function f over a certain domain $\mathcal{D}$. The procedure consists of three steps: the first in which the domain $\mathcal{D}$ of the function is partitioned into several sub-domains; the second in which f is approximated in each of the sub-domains by regular functions following the condition in Expression (1) and a minimum point

of the regular function is obtained; and a third step, in which the minimum points computed in the second step are considered and the best among them is chosen as the output. This procedure can be further strengthened by using lower bounds to check if a given sub-domain can possibly lead to a solution better than any found thus far. The approximation of f by regular functions g_i is called *data-correcting*, since an easy way of obtaining the regular functions is by altering the data that describe f. A pseudocode of the algorithm, which we call DC-G, is provided in Figure 1.

Procedure DC-G
Input: f, $\mathcal{D}$, α.
Output: $x^\alpha \in \mathcal{D}$ such that $f(x^\alpha) \leq \min\{f(x) | x \in \mathcal{D}\} + \alpha$.
Code:

```
1.  begin
2.      create a partition {D_1, ..., D_n} of D;
3.      for each sub-domain D_i
4.      begin
5.          f_i := a lower bound to f(x), x ∈ D_i;
6.          if f_i ≥ bestvalue
7.              continue;
8.          construct a regular function g_i(x) obeying Expression (1);
9.          x_i^α ∈ arg min_{x∈D_i}{g_i(x)};
10.     end;
11.     bestvalue := ∞;
12.     if f(x_i^α) < bestvalue;
13.     begin
14.         x^α := x_i^α;
15.         bestvalue := f(x^α);
16.     end;
17.     return x^α;
18. end.
```

FIGURE 1. A data correcting algorithm for a real-valued function

Lines 5 through 7 in the code carry out the bounding process, and lines 8 and 9 implement the process of computing the minima of regular functions over each sub-domain. These steps are enclosed in a loop, so that at the end of line 10, all the minima of the regular functions are at hand. The code in lines 11 through 16 obtain the best among the minima obtained before. By Theorem 1, the solution chosen by the code in lines 11 through 16 is an α-minimum of f, and therefore, this solution is returned by the algorithm in line 17.

We will now illustrate the data correcting algorithm through an example. The example that we choose is one of a real-valued function of one variable, since these are some of the simplest functions to visualize.

Example 1. (Data correcting on a real-valued function of one variable) Consider the problem of finding an α-minimum of the function f shown in Figure 2.

The function is defined on the domain D and is assumed to be analytically intractable.

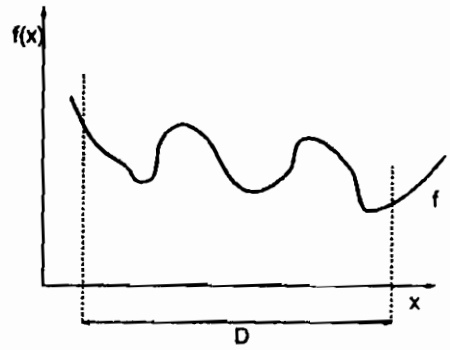


FIGURE 2. A general function f

The data correcting approach can be used to solve the problem above, i.e. of finding a solution $x^\alpha \in D$ such that $f(x^\alpha) \leq \min\{f(x) | x \in \mathcal{D}\} + \alpha$

Consider the partition {D1, D2, D3, D4, D5} of D shown in Figure 3. Let us suppose that we have a regular function g1(x) such that $|g1(x) - f(x)| \leq \frac{\alpha}{2}$, $\forall x \in D1$. Assume also, that x1 is a minimum point of g1(x) in D1. Since this is the best solution that we have so far, we store x1 as an α-minimal solution to f(x) in the domain D1. We then consider the next interval in the partition, D2. We first obtain a lower bound on the minimum value of f(x) on D2. If this bound is larger than f(x1), we ignore this domain and examine domain D3. Let this not be the case in our example. So we construct a regular function g2(x) with $|g2(x) - f(x)| \leq \frac{\alpha}{2}$, $\forall x \in D2$, and find x2, its minimum point over D2. Since $f(x2) \geq f(x1)$ (see Figure 3), we retain x1 as our α-optimal solution over $D1 \cup D2$. Proceeding in this manner, we examine f(x) in D3 through D5, compute regular functions g3(x) through g5(x) for these domains, and compute x3 through x5. In this example, x3 replaces x1 as our α-minimal solution after consideration of D3, and remains so until the end. At the end of the algorithm, x3 is returned as a value of x^α. ◊

There are four points worth noting at this stage. The first is that we need to examine all the sub-domains in the original domain before we return a near-optimal solution using this approach. The reason for this is very clear. The correctness of the algorithm depends on the result in Theorem 1, and this theorem only concerns the *best* among the minima of each of the sub-domains. For instance, in the previous example, if we stop as soon as we obtain the first α-optimal solution x_1 we would be mistaken, since Theorem 1 applies to x1 only over $D1 \cup D2$. the algorithm The second point is that there is no guarantee that the near-optimal solution returned by DC-G will be in the neighborhood of a true optimal solution. There is in fact, nothing preventing the near-optimal solution existing in a sub-domain different from the sub-domain of an optimal solution, as is evident from the previous example. The true minimum of f lies in the domain D5, but DC-G returns x3, which is in D3. The

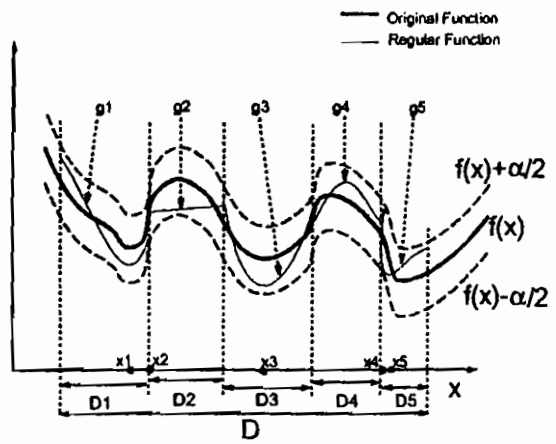


FIGURE 3. Illustrating the Data Correcting approach on f

third point is that the regular functions $g_i(x)$ approximating $f(x)$ do not need to have the same functional form. For instance in Example 1, $g1(x)$ is quadratic, while $g2(x)$ is linear. Finally, for the proof of Theorem 1, it is sufficient for $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ to be a cover of $\mathcal{D}$ (as opposed to a partition).

3. DATA CORRECTING FOR COMBINATORIAL OPTIMIZATION PROBLEMS

The data correcting methodology described in the previous section can be incorporated into an implicit enumeration scheme (like branch and bound) and used to used to obtain near-optimal solutions to NP-hard combinatorial optimization problems. In this section we describe how this incorporation is achieved for a general combinatorial optimization problem. For this purpose, we define a combinatorial optimization problem $\mathcal{P}$ as a collection of instances $\mathcal{I}$. An instance $\mathcal{I}$ consists of a ground set $\mathcal{G} = \{e_1, e_2, \ldots, e_n\}$ of n elements, a cost vector $C_{\mathcal{I}} = (c_1^{\mathcal{I}}, c_2^{\mathcal{I}}, \ldots, c_n^{\mathcal{I}})$ corresponding to the elements in $\mathcal{G}$, a set $\mathcal{S} \subseteq 2^{\mathcal{G}}$ of feasible solutions, and a cost function $f_{\mathcal{I}} : \mathcal{S} \to \Re$. The objective is to obtain a solution, i.e. a member of $\mathcal{S}$ that minimizes the cost function. For example, for an asymmetric traveling salesperson problem (ATSP) instance on a graph $G = (V, A)$, with a distance matrix $D = [d_{ij}]$, we have $\mathcal{G} = A$, $c_{ij}^{\mathcal{I}} = d_{ij}$, $\mathcal{S}$ is the set of all Hamiltonian cycles in G, and $f(s) = \sum_{ij \in s} d_{ij}$ for each $s \in \mathcal{S}$.

Implicit enumeration for combinatorial problems includes two main strategies, namely branching and fathoming. Branching involves partitioning the set of feasible solutions $\mathcal{S}$ into smaller subsets. This is done under the assumption that optimizing the cost function over a more restricted solution space is easier than optimizing it over the whole space. Fathoming involves one of two processes. First, we could compute lower bounds to the value that the cost function can attain over a particular member of the partition. If this bound is not better than the best solution found thus far, the corresponding subset in the partition is ignored in the search for an optimal solution.

The second method of fathoming is by computing the optimum value of the cost function over the particular subset of the solution space (if that can be easily computed for the particular subset). We see therefore that two of the main requirements of the data correcting algorithm presented in the previous section, i.e. partitioning and bounding, are automatically taken care of for combinatorial optimization problems by implicit enumeration. The only other requirement that we need to consider is that of obtaining regular functions approximating $f_{\mathcal{I}}$ over subsets of the solution space.

Notice that the cost function $f_{\mathcal{I}}(s)$ is a function of the cost vector C. So if the values of the entries in C are changed, $f_{\mathcal{I}}(s)$ undergoes a change as well. Therefore, cost functions corresponding to polynomially solvable special cases can be used as "regular functions" for combinatorial optimization problems. Also note that for the same reason, the accuracy parameter can be compared with a suitably defined distance measure between two cost vectors, (or equivalently, two instances). Consider a subproblem in the tree obtained by normal implicit enumeration. The problem instance that is being evaluated at that subproblem is a restricted version of the original problem instance, i.e., it evaluates the cost function of the original problem instance for a subset $\mathcal{S}_k$ of the original solution space $\mathcal{S}$. If we alter the data of the problem instance in a way that the altered data corresponds to a polynomially solvable special case, while guaranteeing that the cost of an optimal solution to the altered problem in $\mathcal{S}_k$ is not more that an acceptable amount higher than the cost of an optimal solution to original instance in $\mathcal{S}_k$, then the altered cost function can be considered to be a regular approximation of the cost function of the original instance in $\mathcal{S}_k$.

For combinatorial optimization problems, let us define a *proximity measure* $\rho(\mathcal{I}_1, \mathcal{I}_2)$ between two problem instances $\mathcal{I}_1$ and $\mathcal{I}_2$, as an upper bound for the difference between $f_{\mathcal{I}_1}(s_1^*)$ and $f_{\mathcal{I}_1}(s_2^*)$, where s_1^* and s_2^* are optimal solutions to $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. The following lemma shows that the *hamming distance* measure between the cost vectors of the two instances is a proximity measure when the cost function is of the sum type or the max type.

Lemma 2. *If the cost function of an instance $\mathcal{I}$ of a combinatorial optimization problem is of the* sum *type, (i.e. $f_{\mathcal{I}}(s) = \sum_{e_k \in s} c_k^{\mathcal{I}}$) or the* max *type, (i.e. $f_{\mathcal{I}}(s) = \max_{e_k \in s} c_k^{\mathcal{I}}$) then the measure*

$$\rho(\mathcal{I}_1, \mathcal{I}_2) = \sum_{e_i \in \mathcal{G}} |c_i^{\mathcal{I}_1} - c_i^{\mathcal{I}_2}| \tag{2}$$

between two instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of the problem is an upper bound to the difference between $f_{\mathcal{I}_1}(s_1^*)$ and $f_{\mathcal{I}_1}(s_2^*)$, where s_1^* and s_2^* are optimal solutions to $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively.

Proof. We will prove the result for sum type cost functions. The proof for max type cost functions is similar.

For sum type cost functions, it is sufficient to prove the result when the cost vectors $C_{\mathcal{I}_1}$ and $C_{\mathcal{I}_2}$ differ in only one position. Let $c_k^{\mathcal{I}_1} = c_k^{\mathcal{I}_2}$ for $k = 1, 2, \ldots, j-1, j+1, \ldots, n$, and $c_j^{\mathcal{I}_1} \neq c_j^{\mathcal{I}_2}$. Consider any solution $s \in \mathcal{S}$. There are two cases to consider:

- $e_j \in s$: In this case, $|f_{\mathcal{I}_1}(s) - f_{\mathcal{I}_2}(s)| = \sum_{e_k \in S} |c_k^{\mathcal{I}_1} - c_k^{\mathcal{I}_2}| = |c_j^{\mathcal{I}_1} - c_j^{\mathcal{I}_2}|$.

- $e_j \notin s$: In this case it is clear that $f_{\mathcal{I}_1}(s) = f_{\mathcal{I}_2}(s)$.

Therefore, $|f_{\mathcal{I}_1}(s) - f_{\mathcal{I}_2}(s)| \leq \sum_{e_i \in \mathcal{G}} |c_i^{I_1} - c_i^{I_2}| = \rho(\mathcal{I}_1, \mathcal{I}_2)$ for any solution $s \in \mathcal{S}$, which automatically implies that $\rho(\mathcal{I}_1, \mathcal{I}_2)$ as defined in the statement of Lemma 2 is an upper bound for the difference between $f_{\mathcal{I}_1}(s_1^*)$ and $f_{\mathcal{I}_1}(s_2^*)$, where s_1^* and s_2^* are optimal solutions to $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. □

One way of implementing the data correcting step for a NP-hard problem instance $\mathcal{I}$ is the following. (We illustrate this step in Example 3 for the ATSP.) We construct a polynomially solvable relaxation $\mathcal{I}_L$ of the original instance (for the ATSP, this can be an assignment problem with the same distance matrix), and obtain an optimal solution x_L to $\mathcal{I}_L$. Note that x_L need not be feasible to $\mathcal{I}$. We next construct the best solution x to $\mathcal{I}$ that we can, starting from x_L. (For the ATSP, this corresponds to a patching operation to obtain a tour from an assignment.) We also construct an instance $\mathcal{I}_C$ of the problem, which will have x as an optimal solution. The proximity measure $\rho(\mathcal{I}, \mathcal{I}_C)$ then is an upper bound to the difference between the costs of x and of an optimal solution to $\mathcal{I}$. $\mathcal{I}_C$ is called a *correction* of the instance $\mathcal{I}$.

The following example illustrates this technique for an instance of the ATSP.

Example 2. (Illustrating data correcting step on a ATSP instance) Consider the 6-city ATSP instance with the distance matrix $\mathbf{D} = [d_{ij}]$ shown below. (This corresponds to $\mathcal{I}$.)

D	1	2	3	4	5	6
1	-	10	16	19	25	22
2	19	-	10	13	13	10
3	10	28	-	22	16	13
4	19	25	13	-	10	19
5	16	22	19	13	-	11
6	13	22	15	13	10	-

If we allow subtours in solutions to the ATSP, we get the classic assignment problem relaxation. Solving the assignment problem on D, using the Hungarian method, we get the following reduced distance matrix $\mathbf{D^H} = [d_{ij}^H]$. (The assignment problem with the distance matrix D corresponds to $\mathcal{I}_L$.)

$\mathbf{D^H}$	1	2	3	4	5	6
1	-	0	6	7	16	12
2	9	-	0	1	4	0
3	0	18	-	10	7	3
4	8	14	2	-	0	8
5	5	11	8	0	-	0
6	2	11	4	0	0	-

This leads to a solution with two cycles (1231) and (4564) (corresponding to x_L). Using patching techniques (see for example [9]), we obtain a solution (1245631) (corresponding to x). Notice that (1245631) would be an optimal solution to the assignment problem if d_{24}^H and d_{63}^H had been set to zero in $\mathbf{D^H}$, and that would have been the situation, if d_{24} and d_{63} were initially reduced by 4 and 1 respectively, i.e.

if the distance matrix in the original ATSP instance was $\mathbf{D^P}$ defined below. (This corresponds to $\mathcal{I}_C$.)

$\mathbf{D^P}$	1	2	3	4	5	6
1	-	10	16	19	25	22
2	19	-	10	9	13	10
3	10	28	-	22	16	13
4	19	25	13	-	10	19
5	16	22	19	13	-	11
6	13	22	14	13	10	-

Therefore $\mathbf{D^P}$ is the distance matrix of the correction of the instance with distance matrix D. The proximity measure $\rho(\mathbf{D}, \mathbf{D^P}) = \sum_{i=1}^{6} \sum_{j=1}^{6} |d_{ij} - d_{ij}^P| = |d_{24} - d_{24}^P| + |d_{63} - d_{63}^P| = 4 + 1 = 5$. ◊

A proximity measure is an upper bound to the difference between the costs of two solutions for a problem instance, so the stronger the bound, the better would be the performance of any enumeration algorithm dependent on such bounds. It is possible to obtain stronger performance measures for ATSP instances, for example

$$\rho_1(\mathbf{D}, \mathbf{D^P}) = \min \left\{ \sum_{i=1}^{n} \max_{1 \le j \le n} |d_{ij} - d_{ij}^P|, \sum_{j=1}^{n} \max_{1 \le i \le n} |d_{ij} - d_{ij}^P| \right\} \tag{3}$$

is a better proximity measure than the one defined in (2). However, the proximity measure ρ has an interesting advantage. Consider the ATSP instance in Example 3. The cost of patching the solution (1231)(4564) to (1245631), i.e. $d_{24} + d_{63} - d_{23} - d_{64}$ is exactly the same as the value of $\rho(\mathbf{D}, \mathbf{D^P})$. This means that we can find out the value of this proximity measure as a by-product of computing the best patching. This is likely to save execution times in any implementation of data correcting algorithms.

The similarity of the data correcting step described above (and illustrated in Example 3) to fathoming rules used in branch and bound implementations makes it convenient to incorporate data correcting in the framework of implicit enumeration. Figure 4 presents the pseudocode of a recursive version of branch and bound incorporating data correcting. The initial input to this procedure is the data for the original instance $\mathcal{I}$, the feasible solution set $\mathcal{S}$, any solution $s \in \mathcal{S}$, and the accuracy parameter α. Notice that the fathoming rule defined in lines 6 through 10 in the pseudocode is the data correcting step discussed earlier in this section.

The algorithm described in Figure 4 is a prototype. We have not specified how the lower bound is to be computed, or which solution to choose in the feasible region, or how to partition the domain into sub-domains. These are details that vary from problem to problem, and are an important part in the engineering aspects of the algorithm. Note that this is just one of many possible ways of implementing the data correcting algorithm.

We next illustrate the data-correcting algorithm on an instance of the ATSP.

Example 3. (Performance of the data correcting algorithm on an ATSP

Algorithm DC
Input: $\mathcal{I}$, $\mathcal{S}$, α.
Output: $x^\alpha \in \mathcal{S}$ such that $f_{\mathcal{I}}(x^\alpha) \leq \min\{f_{\mathcal{I}}(x) | x \in \mathcal{S}\} + \alpha$.
Code:

```
1.  begin
2.      s := a solution in S;
3.      lb := a lower bound on the value of f_I(x) over S;
4.      if f_I(s) = lb
5.          return s;
6.      compute an optimal solution s_L to a polynomially solvable
        relaxation I_L to I;
7.      construct a solution s to I starting from s_L;
8.      construct an instance I_C that has s as an optimal solution;
9.      if ρ(I_C, I) ≤ α
10.         return s;
11.     else
12.     begin
13.         partition S into subsets S_1 and S_2;
14.         s_1 := DC(I, S_1, α);
15.         s_2 := DC(I, S_2, α);
16.         return the better solution among s_1 and s_2;
17.     end;
18. end.
```

FIGURE 4. A data correcting algorithm for a combinatorial optimization problem with minimization objective.

instance) Consider the 8-city ATSP instance with the distance matrix $\mathbf{D} = [d_{ij}]$ shown below. (This example was taken from [1], p. 381).

D	1	2	3	4	5	6	7	8
1	-	2	11	10	8	7	6	5
2	6	-	1	8	8	4	6	7
3	5	12	-	11	8	12	3	11
4	11	9	10	-	1	9	8	10
5	11	11	9	4	-	2	10	9
6	12	8	5	2	11	-	11	9
7	10	11	12	10	9	12	-	3
8	7	10	10	10	6	3	1	-

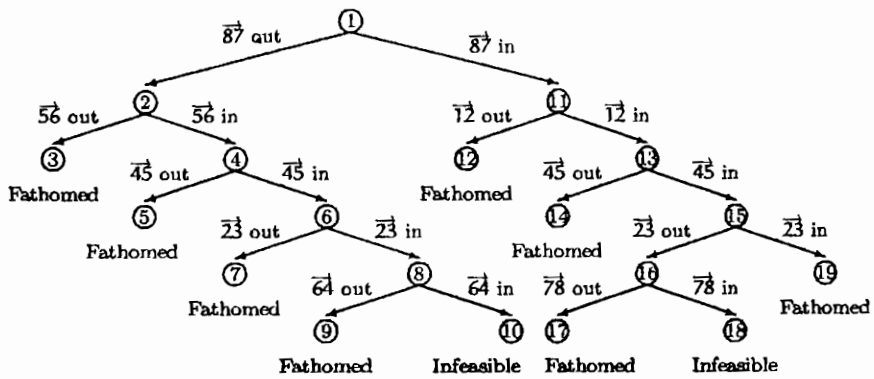
We use

- the proximity measure ρ (see Expression (2)) for data correction,
- the assignment algorithm to compute lower bounds for subproblems,
- a patching algorithm to create feasible solutions, and compute proximity measures, and
- the patched solution derived from the assignment solution as a feasible solution in the domain.

In this example, we branch on the least cost edge involved in the patching operation. (This branching rule is chosen for illustration purposes only, other more efficient branching choices are of course possible.)

The polynomially solvable special case that we consider is the set of all ATSP instances for which the assignment procedure gives rise to a cyclic permutation.

Using the branching rule described above, depth-first branch and bound generates the enumeration tree of Figure 5. The nodes are labelled according to the order in which the problems at the corresponding nodes were evaluated.



Subproblem at node	Upper bound	Lower bound	Assignment solution	Patched tour	Cost of patching	Revised bound
1	∞	17	(1231)(4564)(787)	(123786451)	9	26
2	26	21	(123781)(4564)	(123786451)	5	26
3	26	26	Fathomed by bounds		–	26
4	26	21	(123781)(4564)	(123785641)	9	26
5	26	29	Fathomed by bounds		–	26
6	26	21	(123781)(4564)	(124563781)	10	26
7	26	29	Fathomed by bounds		–	26
8	26	21	(123781)(4564)	(123784561)	13	26
9	26	34	Fathomed by bounds		–	26
10	–	–	Fathomed due to infeasibility		–	26
11	26	17	(1231)(4564)(787)	(187456231)	15	26
12	26	28	Fathomed by bounds		–	26
13	26	17	(1231)(4564)(787)	(123564871)	19	26
14	26	30	Fathomed by bounds		–	26
15	26	17	(1231)(4564)(787)	(126458731)	19	26
16	26	25	(12631)(454)(787)	(126458731)	11	26
17	26	32	Fathomed by bounds		–	26
18	–	–	Fathomed due to infeasibility		–	26
19	26	30	Fathomed by bounds		–	26

FIGURE 5. Branch and bound tree for the instance in Example 3

Since the cost of patching equals the value of ρ, we can now evaluate the performance of data correcting on this example. If the allowable accuracy parameter α is set to 0, then the enumeration tree constructed by DC will be identical to that in Figure 5 and evaluate 19 subproblems. But if the value of α is set to 5, then enumeration along the branch "$\overrightarrow{87}$ out" stops at node 2. In that case, DC evaluates only 11

subproblems. On the other hand, if α is set to 10, DC only evaluates one subproblem (corresponding to node 1). $\Diamond$

The previous example shows that the data correcting algorithm can be a very attractive alternative to branch and bound algorithms. In the next section we report experiences of the performance of the data correcting algorithm on ATSP instances from the TSPlib.

4. Computational Experience with ATSP Instances

In this section we demonstrate the effectiveness of the data correcting algorithm on some benchmark ATSP instances from TSPLIB [11]. TSPLIB has twenty seven ATSP instances, out of which we have chosen twelve which could be solved to optimality within five hours using a branch and bound algorithm. Eight of these belong to the 'ftv' class of instances, while four belong to the 'rbg' class. We implemented the data correcting algorithm in C and ran it on a Intel Pentium based computer running at 666MHz with 128MB RAM.

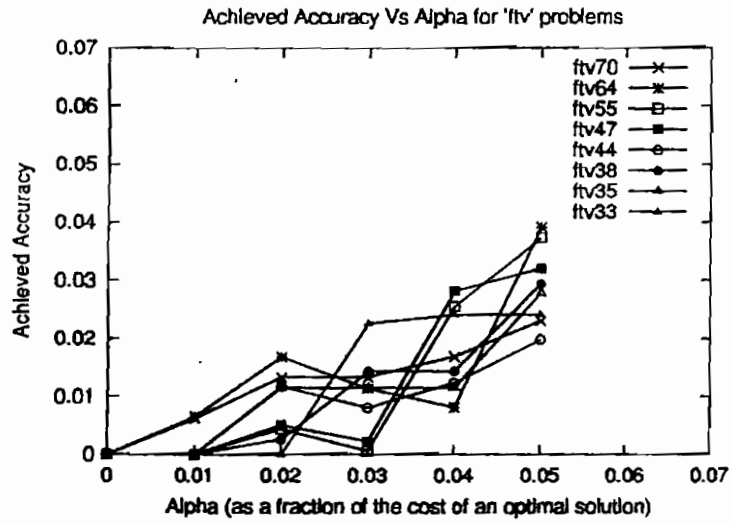


FIGURE 6. Accuracy achieved versus α for ftv instances

The results of our experiments are presented graphically in Figures 6 through 9. In computing accuracies, (Figures 6 and 8) we have plotted the accuracy and deviation of the solution output by the data correcting algorithm from the optimal (called 'achieved accuracy' in the figures) as a fraction of the cost of an optimal solution to the instance. We observed that for each of the twelve instances that we studied, the achieved accuracy is consistently less than 80% of the pre-specified accuracy.

There was a wide variation in the CPU time required to solve the different instances. For instance, ftv70 required 17206 seconds to solve to optimality, while rbg323 required just 5 seconds. Thus, in order to maintain uniformity while demonstrating the variation in execution times with respect to changes in α values, we represented the execution times for each instance for each α value as a percentage of the execution

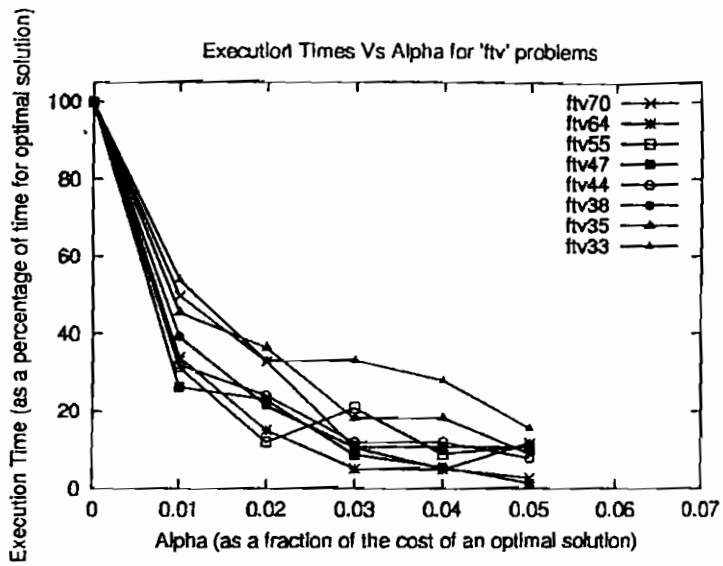


FIGURE 7. Variation of execution times versus α for ftv instances

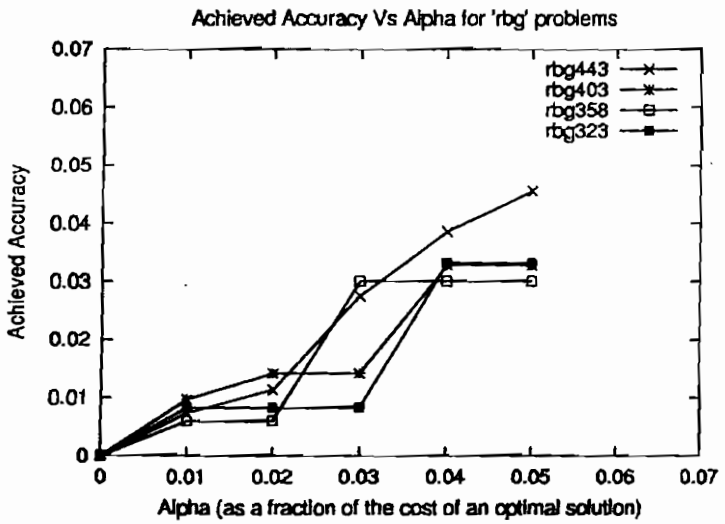


FIGURE 8. Accuracy achieved versus α for rbg instances

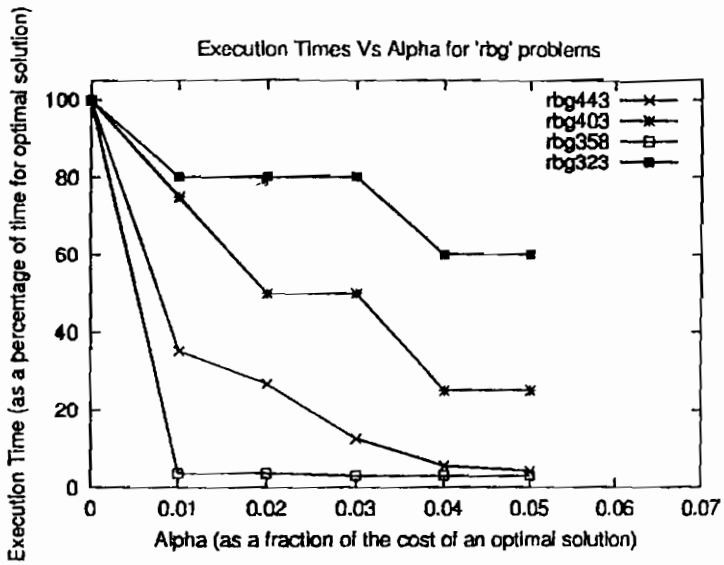


FIGURE 9. Variation of execution times versus α for rbg instances

time required to solve that instance to optimality. Notice that for all the ftv instances when α was 5% of the cost of the optimal solution, the execution time reduced to 20% of that required to solve the respective instance to optimality. The reduction in execution times for rbg instance was equally steep, with the exception of rbg323 which was in any case an easy instance to solve.

5. SUMMARY AND DISCUSSIONS

In this paper we provide an introduction to the concepts of data correcting, a method in which our knowledge of polynomially solvable special cases in a given problem domain is utilized to obtain near-optimal solutions with pre-specified performance guarantees within short execution times. The algorithm makes use of the fact that even if the cost of an optimal solution to a given instance is not known, it is possible to compute a bound on the cost of the solution based on the cost of an optimal solution to another instance. (This fact is proved for a single variable real-valued function in Section 2 of the paper.)

In Section 2, we describe the data correcting process on a single variable real-valued function. Most of the terminology used in data correcting is defined in this section. We also provide a pseudocode for a data correcting algorithm for a general real valued function and an example demonstrating the algorithm. In Section 3, we show how the idea of data correcting can be used to solve combinatorial optimization problems. It turns out that it fits nicely into the framework of branch and bound. We also provide a pseudocode for an algorithm applying data correcting on a combinatorial optimization problem with min-sum objective, and show, using an example, how the algorithm would work on an asymmetric traveling salesperson problem. In Section 4 we describe our computational experience with benchmark asymmetric traveling salesperson problems from the TSPLIB. We show that the deviation in cost of

the solutions output by our data correcting implementation from the optimal is about 80% of the allowable deviation, and the the time required to solve the problems to 95% optimality is about 20% of the time required to solve that particular problem to optimality.

We have used data-correcting primarily for solving NP-hard combinatorial optimization problems. In particular, we have studied the performance of this algorithm on general supermodular and submodular functions [7], quadratic cost partitioning problems [7], simple plant location problems [8], binary knapsack problems [5], and in this paper, on asymmetric traveling salesperson problems. Thus much research remains to be done on testing the performance of this algorithm on other hard combinatorial problems. All our work on data correcting has implemented this method in the branch and bound framework. However, in recent times, dynamic programming is being used to solve large optimization problems. It would be interesting to try to adapt data correcting to work with dynamic programming algorithms.

In addition to solving combinatorial problems, data correcting can also be used for obtaining near-optimal solutions to problems in the continuous domain (as illustrated in Section 2 in this paper). To the best of our knowledge, there has been no work done in this area. This is thus another vast area of research for this algorithm.

REFERENCES

[1] E. Balas, P. Toth, Branch and bound methods, Chapter 10 in [9].

[2] R.E. Burkyard, Special cases of traveling salesman problem and heuristics, *Acta Mathematica Applicatae Sinica* 6, (1990) 273–288.

[3] M. Ebben, *A Data Correcting Algorithm for the Traveling Salesman Problem*, Master's Thesis, Faculty of Economic Sciences, University of Groningen, The Netherlands, 1996.

[4] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman San Francisco, 1979.

[5] D. Ghosh, B. Goldengorin, The Binary Knapsack Problem: Near-Optimal Solutions with Guaranteed Quality, SOM Research Report, University of Groningen, 2001.

[6] P.C. Gilmore, E.L. Lawler, D.B. Shmoys, Well-solved special cases, Chapter 4 in [9].

[7] B. Goldengorin, G. Sierksma, G.A. Tijssen, M. Tso, The data-correcting algorithm for minimization of supermodular functions. *Management Science* 45, (1999) 1539–1551.

[8] B. Goldengorin, G.A. Tijssen, D. Ghosh, G. Sierksma, Solving the Simple Plant Location Problem Using a Data Correcting Approach, SOM Research Report, University of Groningen, 2001 (*To appear: Journal of Global Optimization*).

[9] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, D.B. Shmoys, (Eds.) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley-Interscience, 1985.

[10] B.M.E. Moret, H.D. Shapiro, *Algorithms from P to NP, Volume 1: Design and Efficiency*, The Benjamin/Cummmins Publishing Company Inc., 1991.

[11] G. Reinelt, TSPLIB 95, `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html`, 1995.

P&QM AREA, INDIAN INSTITUTE OF MANAGEMENT, AHMEDABAD, INDIA,
E-mail address: `diptesh@iimahd.ernet.in`

FACULTY OF ECONOMIC SCIENCES, UNIVERSITY OF GRONINGEN, THE NETHERLANDS.
E-mail address: `{B.Goldengorin, G.Sierksma}@eco.rug.nl`