Control and Cybernetics

vol. **35** (2006) No. 3

The corridor method: a dynamic programming inspired metaheuristic

by

Moshe Sniedovich[1] **and Stefan Voß**[2]

[1]Department of Mathematics and Statistics
The University of Melbourne, Australia
[2]Institut für Wirtschaftsinformatik
Universität Hamburg
D - 20146 Hamburg, Germany
e-mail: m.sniedovich@ms.unimelb.edu.au, stefan.voss@uni-hamburg.de

Abstract: This paper presents a dynamic programming inspired metaheuristic called *Corridor Method*. It can be classified as a method-based iterated local search in that it deploys method-based neighborhoods. By this we mean that the search for a new candidate solution is carried out by a fully-fledged optimization method and generates a global optimal solution over the neighborhood. The neighborhoods are thus constructed to be suitable domains for the fully-fledged optimization method used. Typically, these neighborhoods are obtained by the imposition of exogenous constraints on the decision space of the target problem and therefore must be compatible with the optimization method used to search these neighborhoods. This is in sharp contrast to traditional metaheuristics where neighborhoods are move-based, that is, they are generated by subjecting the candidate solution to small changes called moves. While conceptually this method-based paradigm applies to any optimization method, in practice it is best suited to support optimization methods such as dynamic programming, where it is easy to control the size of a problem, hence the complexity of algorithms, by means of exogenous constraints. The essential features of the Corridor Method are illustrated by a number of examples, including the traveling salesman problem, where exponentially large neighborhoods are searched by a linear time/space dynamic programming algorithm.

Keywords: metaheuristics, dynamic programming, curse of dimensionality, corridor method, very large neighborhoods, global optimization, move-based, method-based, traveling salesman problem.

1. Introduction

Traditional metaheuristics such as tabu search, simulated annealing, genetic algorithm and iterated local search, were designed as practical tools for dealing with situations where the *Curse of Dimensionality* makes it difficult, perhaps impossible, to apply conventional optimization methods such as linear programming (LP), dynamic programming (DP) and branch and bound (BB) (Voß, 2001). In this sense they can be regarded as alternatives to optimization methods and are therefore particularly useful in applications where the latter are subjected to the Curse of Dimensionality.

In this paper we describe a metaheuristic of a different type. Conceptually, this metaheuristic is not designed to replace existing optimization methods, but rather to support them in a heuristic manner so that they can deal with the *Curse of Dimensionality.* To explain this idea and the reasoning behind it, let us consider the following abstract, very general, optimization problem:

$$\textit{Problem } P(X): \quad z^* := \mathop{\textit{opt}}_{x \in X} f(x) \tag{1}$$

where X is some set and f is a real valued function on X. We shall refer to this problem as the target problem. We intentionally regard the decision space X as a parameter of the problem, indicating that we plan to solve the problem for sets other than X.

The following situation is quite typical: we have at our disposal optimization methods capable of solving *Problem P(X)* in cases where the decision space X is not too large. However, as X increases in size the computational effort required by these methods is rapidly increasing, so much so that practically they cannot handle problems of sizes encountered in large-scale real-world applications. This is not necessarily a manifestation of an inherent deficiency of conventional optimization methods. Rather, it is a reflection of the fact that many practical problems are hard to solve (Garey and Johnson, 1979).

Richard Bellman - the Father of dynamic programming (DP) - coined the phrase Curse of Dimensionality precisely for the description of this common phenomenon (Bellman, 1957). The classical example is DP's treatment of the standard traveling salesman problem (TSP): the size of the DP model for the TSP grows exponentially with the size of the problem, where size is measured by the number of cities to be visited (Balas and Simonetti, 2001).

Neighborhood search methods deal with the above difficulty by considering a sequence of problems $\{\textit{Problem } P(Y^{(j)}) : j = 1, \ldots, k\}$ where $Y^{(j)}$ is a relatively small subset of X, in fact small enough so that $\textit{Problem } P(Y^{(j)})$ can be easily solved by available optimization methods and/or ad hoc procedures. This process generates a sequence of solutions $(y^{(1)}, \ldots, y^{(k)})$ such that $y^{(j)}$ is an optimal solution to $\textit{Problem } P(Y^{(j)})$. The best element of this sequence, namely $y^{(m)}$ such that $f(y^{(m)}) = \textit{opt}\{f(y) : j = 1, \ldots, k\}$, is designated as the pseudo-optimal solution for $\textit{Problem } P(X)$. If $X \subset \overline{Y} := \bigcup_{j=1}^{k} Y^{(j)}$ then clearly

$y^{(m)}$ is an optimal solution of *Problem* $P(X)$.

But typically, even if $\overline{Y}$ is large in absolute terms, it is a relatively small subset of X, hence there is no guarantee that $y^{(m)}$ is a global optimal solution of *Problem* $P(X)$. For this reason neighborhood search methods are regarded as heuristics. In particular, they deploy heuristics to determine how the sets $\{Y^{(j)}\}$ — called *neighborhoods* - should be generated.

There are three fundamental interrelated issues here, namely

– The basic structure of the neighborhoods.

– The size of the neighborhoods.

– The relationship between successive neighborhoods.

Our discussion will focus almost exclusively on the first two issues, namely the basic structure and size of the neighborhoods.

Traditionally, the structure of neighborhoods was determined by very simple *topological* concepts relating a given feasible solution $x \in X$ to very similar feasible solutions via simple local changes or moves. This gave rise to relatively *small homogeneous* neighborhoods. More recently, much more elaborate types of moves have been used in the construction of very large and diverse neighborhoods (Ahuja et al., 2002). We shall refer to neighborhoods of this type, whether large or small, as *move-based neighborhoods.*

In contrast, the metaheuristic described in this paper, called Corridor Method, deploys neighborhoods that are method-based. By this we mean that the basic structure of a neighborhood is determined by the needs and requirements of the optimization method used to search the neighborhood. So if, for example, the solution method we use is DP, then the structure of the neighborhoods will be suitable for a DP treatment.

With this in mind assume that there is an optimization method, call it M, that is capable of solving *Problem* $P(X)$ in cases where X is not too large. If X is too large, we shall use method M to solve *Problem* $P(Y)$, where Y is a small subset of X. The size and structure of Y is determined by the capabilities and limitations of method M. And since typically Y is much smaller than X, to provide a reasonably good solution for *Problem* $P(X)$ it would be necessary to solve *Problem* $P(Y)$ for several, perhaps many, subsets $\{Y^{(j)}\}$ of X.

The rationale for this scheme is simple: if an optimal solution to *Problem* $P(X)$ is an element of a subset Y of X, then any optimal solution to *Problem* $P(Y)$ is also optimal for *Problem* $P(X)$. Of course, the difficulty is that a priori it is not easy to identify such a subset of X.

As implied by the title of this paper, the development of the *Corridor Method* was inspired by DP, or more precisely by attempts to devise ways to deal with the *Curse of Dimensionality* in the context of large scale DP applications. Conceptually, the idea is to construct a *corridor* around the *state trajectory* generated by an incumbent feasible solution to the target problem and use DP to optimize the objective function over this corridor. The optimal solution generated this way over the current corridor is the new incumbent solution for the target problem. This process is repeated until a *fixed point* is reached: the new incumbent is identical to the current incumbent. At this point the procedure either terminates or a new incumbent solution is generated (somehow) and used as a *seed* for a new run of the iterative search process.

The origin of the *Corridor Method* dates back to the early 1970s when attempts were made to cope with the *Curse of Dimensionality* in the context of DP applications to reservoir control problems (see the survey by Yakowitz, 1982). Indeed, a very simple version of the method, called *Discrete Differential Dynamic Programming (DDDP)* had been used extensively to support DP in the solution of large reservoir operation problems (Heidari et al., 1971; Chow et al., 1975; Yakowitz, 1982).

As indicated in this discussion, the same idea can be used in conjunction with other fully-fledged optimization methods, such as LP and BB, except that the corridor is constructed around the incumbent solution itself rather than around the *state trajectory* it generates. The key here is to generate the corridors by the imposition of *exogenous constraints* on the decision space X of the target problem. Needless to say, these constraints should be compatible with the method used to optimize the objective function over the corridors.

The main objective of this paper is to describe the essential ingredients of this metaheuristic and discuss its relationship to other local search methods. For this purpose it is instructive to take a quick look at the well-established classical *iterated local search method.* This will set the framework for a formal description of move-based neighborhoods, method-based neighborhoods and the *Corridor Method.*

The rest of the paper is organized as follows. After presenting some background from the area of metaheuristics (Sections 2-4 including a distinction between move-based and method-based neighborhoods) we define the *Corridor Method* more formally (Section 5). Section 6 presents some examples to underline our ideas, while Section 7 indicates the applicability of the method to a wide range of combinatorial optimization problems. The paper closes with a short discussion and some conclusions

2. Iterated local search method

Assume that in the framework of *Problem* $P(X)$ there is a map $\mathcal{N}$ from X to the power set of X such that for each set $\mathcal{N}(x)$ is a non-empty strict subset of X. We call $\mathcal{N}$ the neighborhood function and $\mathcal{N}(x)$ the neighborhood of x. Also assume that there is a map Π from X to itself such that $\Pi(x) \neq x, \forall x \in X,$. We call $x' = \Pi(x)$ a *perturbation* of x.

The iterated local search method works as follows (Voß, 2001; Hoos and Stützle, 2004):

```
**Start:**  Set $j = 1$ and select $x^{(1)} \in X$ .
            $x^* = \arg \underset{y \in \mathcal{N}(x^{(j)})}{opt} f(y)$
```

```
Iterate:  Repeat until (Termination Condition)
          If $f(x^*) = f(x^{(j)})$ set $x^{(j+1)} = \Pi(x^*)$ else set $x^{(j+1)} = x^*$
          Set $j = j + 1$
```

where *Termination Condition* is a rule that determines when the iterative procedure should be terminated. For ease of exposition we assume that $x^{(j)} \in \mathcal{N}(x^{(j)})$.

Let $x = (x^{(1)}, \ldots, x^{(k)})$ denote the sequence of x values generated by this procedure and consider a pair of two consecutive elements of this sequence, say $(x^{(j)}, x^{(j+1)})$. There are two possibilities with regard to the relationship between these two solutions: if $x^{(j)}$ is an optimal solution for *Problem* $P(\mathcal{N}(x^{(j)}))$ then $x^{(j+1)}$ is a perturbation of $x^{(j)}$, otherwise it is an optimal solution to *Problem* $P(\mathcal{N}(x^{(j)}))$. Whenever Π is called upon to resolve a fixed-point situation, a new seed $x^{(j+1)} = \Pi(x^{(j)})$ is generated for the local search process.

Three heuristics are embedded then within the iterated local search method:

$\mathcal{N}$: for generating the neighborhoods for the local search method.

Π : for generating new seeds to get out of fixed points.

Termination Condition : for terminating the process.

Our discussion will focus on $\mathcal{N}$.

3. Move-based neighborhoods

Traditionally, the neighborhoods associated with the iterated local search method, indeed also with most other metaheuristics, were based on the elementary move concept. Roughly speaking, an elementary move is a very small change in the structure of a given feasible solution $x \in X$. The most popular elementary moves are *k-opt*, *swap*, *insertion*, and *bit-flip* (Voß, 2001). We distinguish between two types of move-based neighborhoods, namely *single-move* based neighborhoods and *multiple-move* based neighborhoods.

Suppose that we have a given set of elementary moves. Let $Move(x)$ denote the single-move based neighborhood of x pertaining to this set. That is, let $Move(x)$ denote the subset of X consisting of feasible solutions that can be obtained from x by a single elementary move. More formally,

$$Move(x) := \{y \in X : (x, y) \in eMoves\} \;\; x \in X \tag{2}$$

where $eMoves$ is the relation on X representing the elementary moves under consideration. By definition, $eMoves$ is a subset of $X \times X$ such that $(x, y) \in eMoves$ if and only if y can be obtained from x by a *single* elementary move.

The basic characteristics of neighborhoods of this type is that they are relatively *small* compared to X and quite *homogeneous*. That is, $Move(x)$ is a relatively small subset of X, the neighborhood of x, consisting of solutions that are fairly similar to x.

Let $Moves(x)$ denote the multiple-move based neighborhood of x associated with the given set of elementary moves. Informally speaking, this set consists of all elements of X that can be obtained from x by successive applications of elementary moves, subject to some *regularity conditions*. It is necessary to impose these regularity conditions because otherwise we could have $Moves(x) = X, \forall x \in X$, which in view of our attempt to make the neighborhoods much smaller than X, is highly undesirable. Furthermore, these regularity conditions may have to be cleverly formulated so that even though $Moves(x)$ is very large compared to $Move(x)$, *Problem* $P(Moves(x))$ can be easily solved.

For the purposes of our discussion it suffices to indicate that $Moves(x)$ is constructed so that (a) it is a relatively small subset of X, (b) it contains, and is much larger than, $Move(x)$, and (c) it is much more diversified than $Move(x)$.

It should be pointed out that the origin of the concept of *elementary move* is very strongly associated with *permutation problems*, namely problems where X is a *permutations* set. Indeed, the three most popular elementary moves *(k-opt, swap, insertion)* are all very much permutation oriented. The traveling salesman problem (TSP) is the prime example of such problems. While it is true that the class of permutation oriented optimization problems plays an extremely important role in combinatorial optimization, there are many other important classes, e.g. knapsack problems, resource allocation problems, inventory problems, production problems and so on. The question therefore arises: how do you construct large, diverse neighborhoods for such problems, where there are no obvious elementary moves?

To be effective, the structure of neighborhoods must be in tune with the structure of the *objective function* f of *Problem* $P(X)$. Since the main purpose of the local search is to improve the value of the objective function, neighborhoods should be constructed with this purpose in mind. But by their very nature elementary moves are completely oblivious to this consideration. Therefore, intelligent local searches whose neighborhoods are based on elementary moves must use some other tools to cope with this issue.

In summary, the traditional, most popular paradigm for the construction of neighborhoods for local searches is very much move-based and is strongly tied to permutation problems. And it should not come as a surprise that DP can be used within the framework of iterative local search processes whose neighborhoods are moved-based (Potts and van de Velde, 1995).

4. Method-based neighborhoods

As attempts are being made to increase the size of neighborhoods deployed by metaheuristics in general and the iterated local search method in particular, it becomes necessary to conduct the local search by means of *fully-fledged optimization methods*. This seems to be a natural development in the history and evolution of metaheuristics.

In this conceptual framework *method-based neighborhoods* constitute a new breed of neighborhoods whose basic structure is determined by the (fully-fledged) optimization method used to search the neighborhoods. Indeed, the basic idea behind method-based neighborhoods is extremely simple, intuitive and powerful: if we are to obtain an exact global optimal solution to *Problem* $P(Y)$ for a large neighborhood Y, then we have to use a fully-fledged optimization method for this purpose. This being the case, the essential structure of Y must be determined, indeed dictated, by the capabilities and limitations of the optimization method used.

The fundamental question is therefore as follows: suppose that we plan to use method M to solve *Problem* $P(Y)$, where Y is a subset of X. How should we construct Y so that (a) Y is much smaller than X but still quite large in absolute terms, and (b) *Problem* $P(Y)$ can be easily solved by method M? Needless to say, this question is raised in the framework of situations where the essential structure of X is compatible with method M, but X is too large.

One obvious answer to this fundamental question is as follows: Y can be constructed by *constraintification*, namely by the imposition of *exogenous constraints* on *Problem* $P(X)$. By necessity, these constraints must be in tune not only with the structure of *Problem* $P(X)$ but also with the capabilities and limitations of method M. In particular, the exogenous constraints should be formulated so as to provide explicit control not only on the size of Y but also on the complexity of the algorithm deployed by method M to solve *Problem* $P(Y)$. In fact, strictly speaking, the size of Y as such is not an issue here, what is important is that method M is capable of solving *Problem* $P(Y)$ efficiently. In this sense the size of Y is merely a proxy for the complexity of the algorithm used by method M to solve *Problem* $P(Y)$.

The following example illustrates some of the conceptual and technical issues associated with the construction of method-based neighborhoods.

Example 4.1

Consider the standard TSP and let $x' = (x'_1, \dots, x'_n)$ be a feasible tour, where $x'_0 = x'_{n+1} = 0$ denotes the home city. What would be a suitable neighborhood around this tour?

The first thing to observe is that in the context of our discussion this question is meaningless because there is no reference to any particular optimization method to be used to search the neighborhoods. So suppose that we rephrase the question thus: What would be a suitable neighborhood around x' given that $M = DP$?

To answer this question we have to identify the reasons why the DP formulation of the TSP is subjected to the Curse of Dimensionality. So recall that this formulation is as follow:

TSP_DP:

$$z^* := \min_{x_1,\dots,x_n} \sum_{j=1}^{n} d(c_j, x_j) + d(c_{n+1}, 0)\ ,\ J := \{0, 1, \dots, n\} \tag{3}$$

subject to

$$\begin{aligned} c_1 &= 0\ ;\ v_1 = \{1, \dots, n\} && (4)\\ c_{j+1} &= x_j\ ,\ j = 1, \dots, n && (5)\\ v_{j+1} &= v_j \backslash \{x_j\}\ ,\ j = 1, \dots, n && (6)\\ x_j &\in v_j\ ,\ j = 1, \dots, n && (7) \end{aligned}$$

where 0 represents the home city and $d(i, j)$ denotes the distance between city i and city j.

The DP functional equation for this model is as follows:

$$f(v, c) = \min_{i \in v} \{d(c, i) + f(v \backslash \{i\}, i))\}\ ,\ v \subset J, c \in J \backslash v \tag{8}$$

with $f(\phi, c) := d(c, 0), c \in J$, where ϕ denotes the empty set. Here, $f(v, c)$ denotes the length of the optimal subtour over the remaining cities given that we are at city c, still have to visit the cities in set v and return to the home city. By definition, $z^* = f(J, 0)$.

The state variable in this model is then of the form $s = (v, c)$ where v denotes the set of cities that have *not yet* been visited and c denotes the *current* city. Let S denote the set of all such states. Since the problem is of size n (there are n cities to visit, excluding the home city), if each city is directly connected to all other cities, we would have $|S| = n2^{n-1}$. The culprit here is then set v: it can take 2^{n-1} distinct values for any given value of city c. Thus, to control the size of the neighborhoods we need to control the number of values that v can take. It follows then that the neighborhood function $\mathcal{N}$ should be constructed so that it is easy to control the number of feasible values that v can take for any given feasible value of c.

This can be done in many different ways, the most straightforward being the imposition of simple exogenous constraints on v and c requiring that the (ordinal) position of the cities in v must not be too far from the position of city c on the incumbent tour x. To explain how this scheme works consider the tour $x = (1, 6, 4, 8, 3, 7, 2, 5, 10, 9)$ in the context of a TSP of size $n = 10$. On completion of the sub-tour $(1, 6, 4, 8, 3, 7, 2)$ we shall observe the state $s = (\{5, 9, 10\}, 2)$, that is, we shall be in city $c = 2$ and will still have to visit the cities in $v = \{5, 9, 10\}$. The reason why we have to record v is the restriction that each city other than the origin should be visited exactly once. Thus, to satisfy this condition we must know what cities have not been visited yet and the next destination must be one of these cities. If v is empty, we must go to city 0 to complete the tour.

Now, suppose that given an incumbent tour $x = (x_1, \dots, x_n)$ and a positive integer $m > 0$ much smaller than n we impose the following exogenous precedence constraint on the problem:

Exogenous Precedence Constraint:
C_1: Given an incumbent tour $x = (x_1, \dots, x_n)$, for each $j = 1, \dots, n$, city x_j can be visited only *after* all the cities $\{x_1, \dots, x_{j-m-1}\}$, where m is a given integer smaller than n.

For example, if $m = 2$ then in the context of the tour $x = (1, 6, 4, 8, 3, 7, 2, 5, 10, 9)$ when we are in city $c = 2$ we are only interested in instances of v that are subsets of $\{3, 5, 7, 10\}$ and there are only $2^4 = 16$ such subsets. Note that if the current city is $c = x_7 = 2$, then in accordance with C_1 we must have already visited all the cities in $\{x_1, \dots, x_4\} = \{1, 4, 6, 8\}$ but definitely have not yet visited city $x_{10} = 9$.

Thus, subject to C_1 the cardinality of the state space of the DP model will not exceed $n2^{2m}$. Therefore, if m is small, say equal to 3 or 4, the DP algorithm will be able to easily handle very large values of n. Of course, this also means that it might be necessary to apply the DP algorithm to many neighborhoods in order to obtain a good solution for the target problem. This should not come as a surprise, after all the TSP is an NP-hard problem (Garey and Johnson, 1979)

As we show below, the size of the neighborhoods induced by this simple exogenous precedence constraint is *exponential* with n. More specifically, for $m > 2$ we have $|\mathcal{N}(x)| > a^n$, where $a > 1$. The complexity of the DP algorithm is $O(n2^{2m})$ per neighborhood, namely it is *linear* with the size (n) of the target problem.

In summary then, using a very simple constraintification scheme, the search over an extremely large decision space ($|X| = n!$) is carried out over exponentially large neighborhoods ($|\mathcal{N}(x)| > a^n, a > 1$) using a *linear* time/memory DP algorithm (per neighborhood).

The next example illustrates again the point that sometimes it is more convenient to construct neighborhoods not around x itself but around a related object. In the case of DP this object is typically the state trajectory.

Example 4.2

Consider the standard 0-1 knapsack problem. Here a solution x is a binary vector in $\{0, 1\}^n$ where n denotes the number of items available. For instance, suppose that $n = 5$,

$$X = \{x \in \{0, 1\}^5 : 11x_1 + 12x_2 + 13x_3 + 14x_4 + 15x_{15} \leq 40\} \tag{9}$$

and consider the initial guess $x^{(1)} = (1, 1, 1, 0, 0)$. How do we construct a neighborhood around this feasible solution?

As was indicated above, strictly speaking this question is not well defined because any meaningful answer must take into consideration the *method* we plan to use to solve the knapsack problem over some subset Y of X. So let us rephrase the question: How do we construct a neighborhood around this solution for $M = DP$?

For the purposes of this discussion it suffices to indicate that within the DP framework it will be convenient to use neighborhoods around the *state trajectory* rather than around the decision variable x, observing that the state variables in this case are as follows:

$$s_j = b - \sum_{i=1}^{j-1} w_i x_i \ , \ j = 1, \ldots, n+1 \tag{10}$$

where in the context of (9) we have $b = 40$ and $w = (11, 12, 13, 14, 15)$.

Note that the state trajectory generated by the solution $x' = (1, 1, 1, 0, 0)$ is equal to $t = (40, 29, 17, 4, 4, 4)$.

The most simplistic neighborhood in this case seems to be a *corridor* of variable width around the state trajectory, say $C = C_1 \times C_2 \times \cdots , \times C_6$, where $C_i = \{\underline{\sigma}_i, \ldots, \overline{\sigma}_i\}$,

$$\underline{\sigma}_j = \max(0, s_j^{(1)} - \delta) \ ; \ \overline{\sigma}_j = \min(b, s_j^{(1)} + \delta) \ , \ j = 2, \ldots, 6 \tag{11}$$

with $C\_1 = b$ and $\delta > 0$.

For example, for $\delta = 10$ the corridor around $t = (40, 29, 17, 4, 4, 4)$ will be

$$\begin{aligned} C =& \{40\} \times \{19, \ldots, 39\} \times \{7, \ldots, 27\} \times \{0, \ldots, 14\} \times \{0, \ldots, 14\} \\ & \times \{0, \ldots, 14\}. \end{aligned} \tag{12}$$

The parameter δ can be used to control the size of the neighborhoods, observing that the above scheme can be refined by allowing the width of the corridor to vary with the stage variable j.

Note that once a corridor is determined, the set of feasible decisions pertaining to a given state is modified accordingly: a decision that generates a state outside the corridor is regarded as infeasible. Thus, since in our example $s_{j+1} = s_j - w_j x_j$, we require x_j to satisfy the constraint $\underline{\sigma}_{j+1} \leq s_j - w_j x_j \leq \overline{\sigma}_{j+1}$.

The complexity analysis is as follows. In the worst case $|X| \approx 2^n$, whereas the complexity of the DP algorithm is $O(nb)$ for *Problem* $P(X)$. For the neighborhoods we have $|\mathcal{N}(s)| \approx (1+2\delta)^n$ so we should expect $\mathcal{N}(s)$ to be larger than X. This is so because the construction of the neighborhoods outlined above allows the inclusion of many state trajectories that are not feasible with respect to the target problem and which will not be considered by the DP algorithm. In any case, the complexity of the DP algorithm, measured by the size of the state space of a neighborhood, is $O(n(1 + 2\delta))$.

This example is a simple instance of a *multi-dimensional grid model.* It consists of two basic constructs, namely stages and states. The stage variable,

$j = 1, 2, \ldots, n$, is a decision making counter and the state variable s_j describes the state of the system at stage j. For simplicity we assume that all the states are m-vectors whose components are non-negative integers. Let S_j denote the set of feasible values of s_j and let $\mathcal{T}$ denote the set of all feasible *state trajectories*, namely let $\mathbf{I} := \{0, 1, ...\}$ and let $\mathcal{T}$ be the subset of $(\mathbf{I}^m)^n$ whose elements $s \in \mathcal{T}$ satisfy the condition $s_j \in S_j$ for $j = 1, \ldots, n$. Let $s_j^{(i)}$ denote the i-th component of s_j so that $s_j = (s_j^{(1)}, \ldots, s_j^{(m)})$. For simplicity we assume that

$$S_j = \left\{ s_j \in \mathbf{I}^m : \underline{s}_j^{(i)} \leq s_j^{(i)} \leq \overline{s}_j^{(i)} \right\} \tag{13}$$

where $\underline{s}_j^{(i)} \leq \overline{s}_j^{(i)}$ are given parameters. By construction then,

$$|\mathcal{T}| = \prod_{j=1}^{n} |S_j| = \prod_{j=1}^{n} \prod_{i=1}^{m} (1 + \overline{s}_j^{(i)} - \underline{s}_j^{(i)}). \tag{14}$$

A corridor around $s \in \mathcal{T}$ is a subset of $\mathcal{T}$ whose elements are within a prescribed distance from s_j at each stage j. That is,

$$C(s, \Delta) = \left\{ \sigma \in \mathcal{T} : |\sigma_j^{(i)} - s_j^{(i)}| \leq \Delta_j^{(i)} \ , \ 1 \leq j \leq n; 1 \leq i \leq m \right\} \tag{15}$$

where the width of the corridor, Δ, is a control parameter.

By construction then,

$$|C(s, \Delta)| = \prod_{j=1}^{n} \prod_{i=1}^{m} \theta_j^{(i)} \geq \mu^n \ , \ \mu := \min \left\{ \prod_{i=1}^{m} \theta_j^{(i)} : 1 \leq j \leq n \right\} \tag{16}$$

where

$$\theta_j^{(i)} = 1 + \min\{u_j^{(i)}, s_j^{(i)} + \Delta_j^{(i)}\} - \max\{l_j^{(i)}, s_j^{(i)} - \Delta_j^{(i)}\}. \tag{17}$$

Observe that if $\theta_j^{(i)} \geq 2$ for all i and j then $|C(s, \Delta)| > 2^n$.

In short, the multi-dimensional grid model provides a straightforward framework for creating exponentially large neighborhoods.

As indicated above, it should be noted that the term size used in our discussion in relation to X and $\mathcal{N}(x)$ should be interpreted with imagination. It definitely does not always refer to the cardinality of these sets. By the same token, the search over $\mathcal{N}(x)$ does not have to be explicit in nature, so it is not always the case that the neighborhoods have to be explicitly defined nor explicitly enumerated.

Perhaps the best example for such a case is *M=Simplex Method of linear programming.* If we define “size” to be the number of variables in the LP model, then a simple constraintification (setting variables to zero and excluding them from the model) could be used to reduce the size of X to any desirable level. Indeed, such neighborhoods are routinely deployed by the famous column generation method of linear programming (Winston, 1994).

An interesting issue regarding method-based neighborhoods is that *Problem* $P(X)$ may sometimes have two or more distinct formulations pertaining to the same optimization method M. In such cases the choice of exogenous constraints to be deployed for the construction of neighborhoods may depend not only on the solution method used but also on the formulation selected. The following example illustrates this point.

Example 4.3

Consider again the standard TSP, assume that M =*Linear Mixed Integer Programming* and that the exogenous precedence constraint C_1 discussed in Example 4.1 is used to construct neighborhoods. Now, the TSP has many distinct mixed integer linear programming formulations and the question therefore arises which one, if any, can handle this type of constraint. We shall consider two well-known types of formulations and examine how they cope with this constraint.

The first type — Dantzig/Fulkerson/Johnson (DFJ) (1954) — is based on explicit subtour elimination constraints and reads as follows:

TSP_DFJ:

$$z^* := \min_{x\in\{0,1\}^{n\times n}} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \tag{18}$$

subject to

$$\sum_{j=1}^{n} x_{i,j} = 1 \ , \ i=1,\dots,n \ , \ ; \ \sum_{i=1}^{n} x_{i,j} = 1 \ , \ j=1,\dots,n \tag{19}$$

$$SEC(x,s) \ , \ \forall s \in J \tag{20}$$

where J denotes the set of all proper non-empty subsets of $\{1,\dots,n\}$ and $SEC(x,s)$ is a typical *subtour elimination* constraint.

The trouble maker in this formulation is set J in (20): its cardinality is 2^n-2. The following is a common instance of (20):

$$\sum_{i,j\in s} x_{i,j} \le |s|-1 \ , \ \forall s\subset\{1,\dots,n\}, \ 2\le|s|\le n-1. \tag{21}$$

How do we formulate in this framework the exogenous precedence constraint C_1 and how do we use this constraint to control the size of J in (20)? Note that there is no apparent direct way of handling C_1 without the introduction of additional variables.

With this in mind consider the Miller/Tucker/Zemlin (MTZ) (1960) formulation which is based on explicit *sequencing variables*:

TSP_MTZ:

$$z^* := \min_{u\in\Re^n, x\in\{0,1\}^{n\times n}} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \tag{22}$$

$$\sum_{j=1}^{n} x_{i,j} = 1 \ , \ i=1,\dots,n \ , \ ; \ \sum_{i=1}^{n} x_{i,j} = 1 \ , \ j=1,\dots,n \tag{23}$$

$$SVC(x,u) \ , \ \forall s \in J \tag{24}$$

where $SVC(x,u)$ represents a set of no more than n^2 sequencing constraints. The following is a common instance of (24):

$$\begin{array}{l} u_1 = 1 \\ 1 \leq u_j \leq n-1 \ , \ j=2,\dots,n \\ u_j \geq u_i + 1 - (n-1)(1-x_{i,j}) \ , \ \forall i,j \geq 2, i \neq j. \end{array} \tag{25}$$

For obvious reasons we shall refer to the $x_{i,j}$ variables as *assignment variables* and to the u_j variables as *sequencing variables*. The former are binary, the latter are continuous.

This particular model consists then of n continuous variables, n^2 binary variables and roughly $2n + (n-1)^2$ constraints.

Note that with the aid of the sequencing variables $(u'_1,\dots,u'_n)$ it is straightforward to incorporate into this model the exogenous precedence constraint C_1. For instance, this constraint can be formulated as follows:

$$u_i \geq u_j \ \ \forall i,j \text{ such that } |u'_i - u'_j| > m \tag{26}$$

where $(u'_1,\dots,u'_n)$ denotes the feasible instantiation of the sequencing variables associated with a feasible solution to the target problem, observing that in the context of (25) city 1 is the home city.

The impact of the exogenous constraint will thus be to significantly reduce the number of assignment variables in accordance with (25), observing that $u_j > u_i$ implies that $x_{i,j} = 0$. Indeed, (26) instantiates (to zero) more than $(n-m)2/2$ of the n^2 assignment variables $x_{i,j}$. Thus, for say $n = 100$ and $m = 2$, this constraint will instantiate (to zero) more than 4802 out of the $10,000$ assignment variables $(x_{i,j})$. As in the case of the DP formulation, the number of feasible tours contained in the corridors induced by the exogenous constraint is exponential with n.

In short, in addition to tightening the constraints on the sequencing variables $(u'_1,\dots,u'_n)$ via (26), the imposition of the exogenous constraints under consideration also significantly decreases the size of the model by instantiating a large portion of the assignment binary variables.

The conclusion is therefore that sequencing-variables-based formulations of the TSP seem to provide a more direct way to handle the specific exogenous constraint than the subtour-elimination-constraints-based formulations.

The foregoing analysis is also valid for other versions of the MTZ model. Details regarding such formulations and the impact they have on the performance of linear integer programming based algorithms for the TSP and precedence constrained TSP can be found in Sherali and Driscoll (2002).

Extensive experience has been accumulated over the years regarding the type of constraints that existing optimization problems can cope with and the impact of such constraints on the performance of algorithms based on these methods. This experience can facilitate the identification of suitable constraints for the construction of neighborhoods for the *Corridor Method.* The key point to remember is that these constraints depend on both the structure of *Problem* $P(X)$ and the optimization method used to solve this problem, M.

With this in mind let us now have a look at the *Corridor Method* and its DP roots.

5. The Corridor Method

As was indicated already, the origin of the *Corridor Method* can be traced back to the 1970s and to the employment of DDDP to solve large-scale reservoir control problems (Heidari et al., 1971; Chow et al., 1975). Since this predated the birth of the field of metaheuristics it is not surprising that the basic idea behind this method has not been developed any further from a metaheuristic perspective. The objective of this section is then two-fold: first to give this basic idea a DP-free formulation and second to cast it as a formal metaheuristic.

Conceptually, there are two ways to describe the *Corridor Method* with this purpose in mind. The first simply draws attention to the fact that the deployment of sequences of admissible elementary moves is just one way of generating large neighborhoods that are easily searchable by existing optimization methods. The point is that there might be other ways to accomplish this goal. Thus, the *Corridor Method* can be described as an iterated local search method where $\mathcal{N}(x)$ is a relatively large set (compared to single-move based neighborhoods) whose structure and size are compatible with the optimization method operating on it, M.

Another way to describe the *Corridor Method* is to reiterate what we pointed out at the outset, namely: typically *Problem* $P(X)$ can be easily solved by some fully-fledged optimization methods if X is not too large. So, suppose that method M can solve *Problem* $P(Y)$ not for the given decision set ($Y = X$), which is too large, but for decision spaces that are much smaller than X itself, yet still very large in absolute terms. Of course, if subject to the prevailing CPU time/memory resources, the method M is capable of solving *Problem* $P(X)$, then it is not necessary to go through the iterative local search method. In this case M can be applied directly to *Problem* $P(X)$.

The implication is that the *Corridor Method* is designed for situations where method M cannot be applied directly to *Problem* $P(X)$ because X is too large, but it can be applied to *Problem* $P(Y)$ if set Y is much smaller than X. The obvious price we pay for deploying this approach is that in general there is no guarantee that the iterative procedure yields a global optimal solution to *Problem* $P(X)$. This is precisely why the method is regarded as a heuristic.

This preliminary examination suggests that a suitable environment for using the *Corridor Method* should have the following basic characteristics:

– The optimization problem under consideration is 'large'.
– An optimization method is available for efficiently solving smaller instances of the problem.
– It is easy to generate (an initial) feasible solution for the target problem.
– There is an efficient method for generating suitable large neighborhoods around feasible solutions to the problem on which the optimization method can be used.

The first three characteristics are universal in nature and require no further discussion. The fourth is clearly a problem/method oriented feature and must be dealt with on a case-by-case basis. It should be stressed that, as in the case of other metaheuristics, even if these four fundamental conditions hold, in general there is no guarantee that the *Corridor Method* will generate good solutions.

The following is then a schema of the *Corridor Method*:

```
Given:     Problem $P(X)$ and method $M$.
Prepare:   $\mathcal{N}_M$, Termination Condition$_M$, $\Pi_M$.
Start:     Set $j = 1$ and select $x^{(1)} \in X$.
Iterate:   Repeat until (Termination Condition$_M$)
           $x^* = \arg \underset{y \in \mathcal{N}(x^{(j)})}{opt_M} f(y)$
           If $f(x^*) = f(x^{(j)})$ set $x^{(j+1)} = \Pi(x^{(j)})$ else set $x^{(j+1)} = x^*$.
           Set $j = j + 1$.
```

Here we use $\mathcal{N}_M$, rather than $\mathcal{N}$, *Termination Condition*$_M$ rather than *Termination Condition*, Π_M rather than Π, and *opt*$_M$ rather than *opt* as a reminder that these objects are very much dependent on method M.

We mention in passing that although officially the incumbent solution x is not required to be a member of its neighborhood, $\mathcal{N}(x)$, in practice this condition is often satisfied in a natural way so no special measures have to be taken to enforce it. The examples discussed above exhibit this feature. The point is that if $\mathcal{N}$ is constructed such that $x \in \mathcal{N}(x), \forall x \in X$, then any sequence generated by the *Corridor Method* for a given seed is monotonic in the desirable way (increasing if *opt* = max and decreasing if *opt* = min). This implies, among other things, that if an optimal solution exits, then any such sequence must converge.

In summary, the *Corridor Method* is an iterated local search method characterized by the property that the neighborhoods that it deploys are large and their construction is governed by the desire to make these neighborhoods easily searchable by a given optimization method (M). In this sense, the *Corridor Method* can be viewed as a method-based rather than move-based iterated local search.

It should be stressed, though, that in principle there is no inherent conflict between the terms move-based and method-based in that elementary moves can be method oriented. In particular, as we have already mentioned above, the construction of multiple-move based neighborhoods involves the imposition of some regularity conditions on the moves (Voß, 2001; Ahuja et al., 2002). These regularity conditions can depend on method M.

6. Illustrative examples

The objective of the illustrative examples featured in this section is three-fold: first, to illustrate how the *Corridor Method* co-operates with method M. Second, to demonstrate typical runs of the iterative local search procedure that attempts to improve candidate solutions. Third, to illustrate what kind of seeding mechanisms can be used to initiate new runs.

All the examples are of TSPs of size $n = 12$. The distances were generated randomly using a uniform distribution on a 20×20 grid. Small random perturbations were used to make the distances asymmetric. City 1 is designated as the home city. A naive DP algorithm plays the role of method M. The *Nearest Neighbor* heuristic (Winston, 1994) is used to generate the first incumbent tour in the implementation of the *Corridor Method.* Two exogenous constraints are used (jointly) to generate the corridor around the incumbent tour $x \in X$: the one used in Example 4.1, namely C_1, and the following:

C_2: For each $j = 1, \dots, n$, from city x_j we can go only to cities in $\{x_i : |i-j| \leq m, 1 \leq i \leq n, i \neq j\}$.

where m is the parameter used in C_1. These two constraints match each other nicely in that their joint application yields a natural corridor around an incumbent solution.

Example 6.1

This example consists of the TSP whose distance matrix is given in Table 1. The results generated by the *Corridor Method* with $m = 4$ are summarized in Table 2, where $TD(t^{(j)})$ denotes the total distance associated with tour $t^{(j)}$. In this particular case the fixed-point $t = t(5) = t(4)$ is the global optimal solution.

Table 1. Distance matrix for Example 4.3

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	—	8	21	5	24	20	10	13	21	28	9	22
2	7	—	16	10	18	18	15	8	14	24	6	23
3	19	16	—	23	6	7	23	9	8	13	16	20
4	7	13	25	—	27	24	9	15	24	30	15	22
5	23	19	5	27	—	5	24	11	12	9	20	17
6	22	17	7	23	6	—	20	10	13	9	20	15
7	12	15	22	9	22	21	—	15	26	25	18	14
8	12	11	8	15	10	10	14	—	12	17	11	17
9	19	13	8	23	12	13	26	12	—	19	12	27
10	26	26	12	28	8	8	23	18	21	—	26	16
11	9	5	18	15	21	18	19	11	12	27	—	26
12	20	23	21	22	18	15	15	16	25	15	27	—

Table 2. One run of the iterative local search procedure, $m = 4$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$t^{(1)}$	1	4	7	12	6	5	3	9	10	8	11	2	1	122
$t^{(2)}$	1	4	7	12	6	10	5	3	8	9	11	2	1	108
$t^{(3)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105
$t^{(4)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105

Table 3 provides the results for $m = 3$. Note that as in the case of $m = 4$, here the fixed point is a global optimal solution.

Table 4 provides a summary of the results generated for this problem when $m = 2$. Note that in this case the fixed-point $t = t^{(1)} = t^{(2)}$ is not a global optimal solution.

Observe that the choice of the home city could be of significance here because the home city can reach only m cities whereas any city in position j on the current tour such that $n-m > j > m$, can reach $2m$ cities. For example, Table 5 presents the modified distance matrix induced by the tour $t^{(0)}$ generated by the Nearest Neighbor heuristic, with $m = 3$. The home, $c = 1$, can reach only 3 cities by a direct link.

In this sense the home city and its immediate neighbors on any incumbent tour are disadvantaged relative to cities further away from the home city on that tour. One of the implications of this simple observation is that the choice of the home city can be deployed as a *seeding mechanism* to enable the search to get out of *fixed points*. The following example illustrates this point.

Table 3. One run of the iterative local search procedure, $m = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$t^{(1)}$	1	4	7	12	6	5	3	9	8	10	11	2	1	129
$t^{(2)}$	1	4	7	12	6	5	3	10	8	9	11	2	1	121
$t^{(3)}$	1	4	7	12	6	10	5	3	9	8	11	2	1	108
$t^{(4)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105
$t^{(5)}$	1	4	7	12	10	6	5	3	9	8	11	2	1	105

Table 4. One run of the iterative local search procedure, $m = 2$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	1	4	7	12	6	5	3	9	8	2	11	10	1	144
$t^{(1)}$	1	4	7	12	6	5	3	9	8	11	2	10	1	140
$t^{(2)}$	1	4	7	12	6	5	3	9	8	11	2	10	1	140

Table 5. Distance matrix induced by $t^{(0)}$, $m = 3$.

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	—	—	—	5	—	—	10	—	—	—	—	22
2	7	—	16	—	—	—	—	8	14	24	6	—
3	—	16	—	—	6	7	—	9	8	—	—	20
4	7	—	—	—	—	24	9	—	—	—	—	22
5	—	—	5	—	—	5	24	11	12	—	—	17
6	—	—	7	23	6	—	20	—	13	—	—	15
7	12	—	—	9	22	21	—	—	—	—	—	14
8	—	11	8	—	10	—	—	—	12	17	11	—
9	—	13	8	—	12	13	—	12	—	—	12	—
10	26	26	—	—	—	—	—	18	—	—	26	—
11	9	5	—	—	—	—	—	11 12	27	—	—	
12	20	—	21	22	18	15	15	—	—	—	—	—

Example 6.2

Consider the TSP defined by the distance matrix given in Table 6. The first run of the iterated local search using $m = 3$ yields the results depicted in Table 7. So now we need a new seed. To enable the last $m = 3$ cities on the tour interact with the first three cities on the tour, let the new seed be the one obtained from the last tour $t^{(3)}$ by moving the first three cities on the tour, namely $(1, 12, 9)$ to the end of the tour. Then the next run of the iterated local search yields the results presented in Table 8. The new fixed point $t = t^{(1)} = t^{(2)}$ is a global optimal solution. Another, more straightforward way to handle this phenomenon is to use modular arithmetic to define the exogenous constraint C_1, taking into consideration the cyclic nature of feasible tours.

Table 6. Distance matrix of Example 6.2

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	—	18	28	22	23	18	23	18	7	28	15	6
2	16	—	15	15	8	27	7	13	12	13	27	17
3	27	15	—	6	17	31	10	11	23	11	29	24
4	23	13	5	—	16	25	11	5	18	13	22	19
5	24	8	16	17	—	33	7	15	17	10	32	23
6	19	28	31	25	33	—	32	22	20	36	5	12
7	3	8	10	11	6	31	—	13	18	4	30	24
8	19	11	11	6	18	22	12	—	15	17	18	14
9	6	12	23	18	17	21	19	14	—	21	21	10
10	29	11	11	12	9	35	4	15	21	—	32	28
11	15	27	28	23	32	4	30	18	20	34	—	11
12	6	17	25	20	25	14	23	15	10	26	10	—

Table 7. First run of the iterated local search, $m = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	1	12	9	2	7	10	5	8	4	3	11	6	1	126
$t^{(1)}$	1	12	9	2	5	7	10	4	3	8	11	6	1	116
$t^{(2)}$	1	12	9	2	5	7	10	3	4	8	11	6	1	110
$t^{(3)}$	1	12	9	2	5	7	10	3	4	8	11	6	1	110

Table 8. Second run of the iterated local search, $m = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	2	5	7	10	3	4	8	11	6	1	12	9	2	110
$t^{(1)}$	2	5	7	10	3	4	8	11	6	12	1	9	2	100
$t^{(2)}$	2	5	7	10	3	4	8	11	6	12	1	9	2	100

Example 6.3

If the distance matrix is approximately symmetric, it often happens that a non-optimal fix point can be improved by reversing the tour it represents. This observation suggests a simple seeding mechanism. In fact, since computing the length of a reversed tour is easy, it is good policy to do this even before a fixed point is reached. If the reversed tour is shorter, it can be used as a new seed even before a fixed point is reached. As an example of this feature, consider the distance matrix depicted in Table 9.

Like the distance matrices of the previous examples, it is approximately symmetric. The first run of the iterated local search with $m = 3$ yields the results summarized in Table 10.

We now need a new seed. So suppose we reverse the last tour and change the home city from 1 to the $(m+1)$th on that tour. This yields the incumbent tour

(8, 12, 2, 5, 4, 10, 11, 6, 1, 3, 7, 9, 8), whose length is 111. Note that by changing the home city this way, the “old” home city can now directly interact with cities that were out of reach before. The results generated by the iterated local search using this tour as a seed are summarized in Table 11. The fixed point generated by this run is the global optimal solution. We remark in passing that since in all the examples the neighborhood around a candidate tour contains this tour, the sequence of distances $\{TD(t^{(j)})\}$ generated by any run of the iterated search is strictly decreasing, until a fixed point is reached.

Table 9. Distance matrix of Example 6.3

$i\backslash j$	1	2	3	4	5	6	7	8	9	10	11	12
1	—	19	21	15	20	7	23	12	24	5	5	11
2	20	—	24	15	5	28	22	14	22	16	21	10
3	20	26	—	32	27	24	10	15	11	21	16	18
4	14	17	32	—	15	19	32	20	31	11	19	17
5	18	5	29	13	—	27	24	14	25	17	23	12
6	10	27	23	19	27	—	30	19	28	11	10	20
7	23	22	8	32	25	30	—	14	4	23	22	16
8	12	12	12	19	16	20	14	—	12	11	13	5
9	23	21	10	32	25	31	3	11	—	23	23	17
10	4	18	18	12	14	13	24	10	22	—	8	8
11	7	21	18	18	24	9	20	13	22	8	—	14
12	13	10	18	17	10	21	16	5	16	9	12	—

Table 10. First run of the iterated local search, $m = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	TD(t(j))
$t^{(0)}$	1	10	11	6	4	5	2	12	8	3	7	9	1	125
$t^{(1)}$	1	6	11	10	4	5	2	12	8	9	7	3	1	115
$t^{(2)}$	1	6	11	10	4	5	2	12	8	9	7	3	1	115

Table 11. Second run of the iterated local search, $m = 3$

	1	2	3	4	5	6	7	8	9	10	11	12	1	$TD(t^{(j)})$
$t^{(0)}$	8	12	2	5	4	10	11	6	1	3	7	9	8	111
$t^{(1)}$	8	12	2	5	4	10	1	6	11	3	7	9	8	108
$t^{(2)}$	8	12	2	5	4	10	1	6	11	3	7	9	8	108

7. Application areas

By its very nature the *Corridor Method* is designed to provide heuristic support to conventional optimization methods in the context of problems where they face the Curse of Dimensionality. Consequently, application areas of this method are strongly linked to application areas of conventional optimization methods.

In this discussion we examine this issue from two related yet distinct point of views, namely scope and competitiveness. The first examines the *Corridor Method* from the view point of method M. The second assesses to what extent algorithms based on this method are competitive relative to other methods.

7.1. Scope

The scope of the *Corridor Method* is extremely wide because in principle practically any optimization method can be incorporated within its basic paradigm. Indeed, the basic recipe is straightforward: if *Problem* $P(X)$ is too large for method M, break X into smaller subsets $Y^{(j)}$ and use method M to solve these smaller problems. Practically speaking then, the scope of operation of the *Corridor Method* is determined by properly matching optimization methods (M) with problem types (*Problem* $P(X)$) via suitable neighborhoods ($\mathcal{N}(x)$). The construction of neighborhoods is typically guided by the identification of exogenous constraints that can be used effectively to control (reduce) the size of the target problem.

One problem domain that seems to be particularly suitable for the *Corridor Method* is Combinatorial Optimization. Although the *Curse of Dimensionality* thrives in this domain, constraintification-based neighborhoods can often be easily constructed. Furthermore, it is often not difficult to construct these neighborhoods so that they are compatible with existing combinatorial optimization methods such as integer programming (IP), DP and branch and bound (BB).

For the purposes of this discussion it suffices to consider the following three very large classes of combinatorial optimization problems. The neighborhoods are constructed here to suit the case where $M = DP$.

Nonlinear, integer, multi-resource allocation problems:

$$\begin{aligned} \textit{Problem } P1(X): \quad & z^* := \max_x \sum_{j=1}^n b_j(x_j) \ , \ X \subset I^n, I := \{0, 1, \dots\} \\ & \sum_{j=1}^n r_{i,j}(x_j) \leq R_i \ \ i = 1, \dots, m \\ & l_j \leq x_j \leq u_j \ \ j = 1, \dots, n. \end{aligned} \tag{27}$$

where b_j and $r_{i,j}$ are real valued functions.

This is a simple generalization of the knapsack problem featured in Example 4.2. The specific instance where b_j and $r_{i,j}$ are linear yields the very important class of multi-constraints knapsack problems. In the worst case, X can be a very large set:$|X| \approx \prod_{j=1}^n (1 + u_j - l_j)$.

Neighborhoods for the *Corridor Method* can be constructed as corridors around the state trajectory generated by x, observing that here the state variables are multi-dimensional, that is $s_j = (s_j^{(1)}, \dots, s_j^{(m)})$ where

$$s_j^{(i)} := R_i - \sum_{p=1}^{j-1} r_{i,p}(x_p) \ , \ i = 1, \dots, m \, . \tag{28}$$

The size of the neighborhoods can be controlled, parametrically, by the width of the corridor, as outlined in (11) and (15). For example, consider

$$\mathcal{N}(s) = \left\{ y \in (I^m)^n : |y_j^{(i)} - s_j^{(i)}| \le \Delta_j^{(i)}, j = 1, \dots, n \ ; \ i = 1, \dots, m \right\} \tag{29}$$

where are non-negative control parameters. These corridors can be arbitrarily small, depending on the choice of the control parameters, observing that for relatively small $\{\Delta_j^{(i)}\}$ we have

$$|\mathcal{N}(s)| = \prod_{j=1}^{n} \prod_{i=1}^{m} (1 + 2\Delta_j^{(i)}). \tag{30}$$

Hence, these neighborhoods are exponentially large (with respect to n) if they are all strictly positive. The DP model will have n stages and at each stage j there will be at most $\prod_{i=1}^{m}(1 + 2\Delta_j^{(i)})$ feasible states. Thus, the complexity of the DP algorithm is linear with n per neighborhood.

The following is a summary of the (worst case) complexity analysis for this class of problems:

Theorem 7.1 *For Problem P1(X) we have,*

Target Problem	*Neighborhoods*	*DP Model*
$\lvert X\rvert \approx \prod_{j=1}^{n}(1+u_j-l_j)$	$\mathcal{N}(x) \approx \prod_{j=1}^{n}\prod_{i=1}^{m}(1+\Delta_j^{(i)})$	$\sum_{j=1}^{n}\lvert S_j\rvert = \sum_{j=1}^{n}\prod_{i=1}^{m}(1+2\Delta_j^{(i)})$

Note that S_j denotes the state space associated with stage j and that we measure the complexity of the DP model by $\sum_{j=1}^{n}|S_j|$.

Multi-component production problems:

$$\textit{Problem } P2(X): \ z^* := \max_{x,y} \sum_{j=1}^{n} g_j(s_j, x_j, y_j), \ X \subset \left(I^{\,m+1}\right)^n, \ I := \{0, 1, 2, \dots\} \tag{31}$$

subject to

$$\begin{aligned}
s_1 &= (b^{(1)}, \ldots, b^{(m)}) && (32)\\
s_{j+1}^{(i)} &= s_j^{(i)} + x_j^{(i)} - y_j p^{(i)} \ , \ i = 1, \ldots, m; j = 1, \ldots, n && (33)\\
\sum_{i=1}^{m} t^{(i)} x_j^{(i)} &\leq T_j \ , \ j = 1, \ldots, n && (34)\\
y_j &\leq \min \left\{ \frac{sj^{(i)} + x_j^{(i)}}{p^{(i)}} : i = 1, \ldots, m \right\} \ , \ j = 1, \ldots, n && (35)\\
l_j^{(i)} &\leq x_j^{(i)} \leq u_j^{(i)} \ , \ i = 1, \ldots, m; j = 1, \ldots, n && (36)\\
\underline{s}_j^{(i)} &\leq s_j^{(i)} \leq \overline{s}_j^{(i)} \ , \ i = 1, \ldots, m; j = 1, \ldots, n && (37)\\
& x_j \in I^m, y_j \in I \ , \ j = 1, \ldots, n && (38)
\end{aligned}$$

where:

$x_j = (x_j^{(1)}, \ldots, x_n^{(m)})$,

$x_j^{(i)}$ = # of components of type i to be produced in period j (decision variable),

y_j = # of items to be assembled and sold in period j (decision variable),

$s_j^{(i)}$ = # of components of type i in stock at the beginning of period j,

$p^{(i)}$ = # of components of type i required for the production of one item,

$t^{(i)}$ = time (hours) required for the production of one component of type i,

$b^{(i)}$ = # of components of type i in stock at the beginning of period 1,

T_j = total production time (hours) available in period j,

$g_j(s_j, x_j, y_j)$ = net benefit during period j.

The problem can be interpreted as follows: it is required to determine how many items $\{x_j\}$ should be produced in each period over the next n periods so as to maximize the total net benefit over this time horizon. Each item consists of $p^{(i)}$ components of type i. It takes $t^{(i)}$ hours to produce one component of type i and a total of T_j hours of production time is available in period j. The net benefit $g_j(s_j, x_j, y_j)$ during period j depends on the number of items $\{y_j\}$ assembled and sold in period j, the number of components of each type $\{x_j^{(i)}\}$ produced in period j, and the inventory level $s_j^{(i)}$ of component i at the beginning of period j.

We can build corridors around the state trajectories associated with the state variables defined by (33). The result is a neighborhood scheme similar to the one defined by (15).

The decision space is much more complicated here so it is convenient to conduct a very rough (worst case) complexity analysis, assuming for example, that at least two decisions are feasible at each stage and that the control parameters are not degenerate. The results can be summarized as follows:

Theorem 7.2 *For Problem P2(X) we have,*

Target Problem	*Neighborhoods*	*DP Model*
$\lvert X\rvert \geq 2^n$	$\lvert\mathcal{N}(x)\rvert \geq 2^n$	$\sum_{j=1}^{n} \lvert S_j\rvert \leq n\delta^3$

where $\delta = \max\left\{\prod_{i=1}^{m}(1+2\Delta_i^{(j)}) : 1 \leq j \leq n\right\}$.

Additive semi-separable permutation problems:

$$\textit{Problem}\; 3P(X):\;\; z^* := \min_{x_1,\dots,x_n} \sum_{j=1}^{n} d_j(v_j, x_j, x_{j+1})\;,\; X \subset J^n, J = \{1,\dots,n\}. \tag{39}$$

subject to

$$\{x_1,\dots,x_n\} = \{1,\dots,n\} \tag{40}$$
$$x_1 = x_{n+1} = 0 \tag{41}$$
$$v_{j+1} = v_j \bigcup \{x_j\}\;,\; j = 0,\dots,n. \tag{42}$$

This is a simple generalization of the conventional TSP in that it is the sum of terms of the form $d_j(v_j, x_j, x_{j+1})$ rather than of the form $d_j(x_j, x_{j+1})$ as in the case of the TSP. The decision space defined by (39)-(43) is identical to the decision space of the standard TSP.

Observation: Let $\mathcal{N}(x)$ denote the neighborhood defined by the exogenous constraint C_1 described in Example 4.1. Then, for $m > 2$ the neighborhood $\mathcal{N}(x)$ grows exponentially with n. More specifically, for a large n we have $\lvert\mathcal{N}(x)\rvert \geq \sqrt{2}^n$. (Note: for simplicity assume that n is even).

Proof. Assume that $m > 2$ and let x be a feasible tour. Divide x into sections each containing 2 adjacent elements of x, say $(x^{(j)}, x^{(j+1)}), 1 \leq j < n$. By interchanging the elements of each section, we double the number of subtours in each section, so the total number of tours generated this way is equal to $K = 2^{n/2} = \sqrt{2}^n$. Since $m > 2$, all these tours are feasible with respect to the exogenous precedence constraint, hence $\lvert\mathcal{N}(x)\rvert \geq \sqrt{2}^n$. ■

Regarding the complexity of the DP model, as we have already noted in Example 4.1, the cardinality of the state space, S, is not larger than $n2^{2m}$. The following is then a summary of the complexity analysis in this case:

Theorem 7.3 *For Problem P3(X) with $m > 2$ we have*

Target Problem	*Neighborhoods*	*DP Model*
$\lvert X\rvert = n!$	$\lvert\mathcal{N}(x)\rvert \geq \sqrt{2}^n$	$\lvert S\rvert \leq n2^{2m}$

The reference to DP in these cases should not be interpreted as suggesting that DP is the only, or most suitable, candidate for the role of M in the context of these classes of problems. Rather, it should be regarded as a reflection of the fact that in these cases DP is a natural candidate because it is very easy to determine the complexity (size) of the DP model in terms of the parameter used to control the size of the neighborhoods. This means that it is easy to determine — a priori — the complexity of the algorithm used to conduct the local search in each neighborhood (Sniedovich, 1992).

Furthermore, as was indicated at the outset, a very simple version of the Corridor Method, called Discrete Differential DP had been used extensively in the water resources area to support DP in the solution of reservoir operation problems.

However, the competitiveness issue still deserves attention: how good are the solutions generated by the *Corridor Method* in relation to the solutions generated by other methods? We now briefly address this important question while leaving extensive numerical experiments to future research.

7.2. Competitiveness

The quality of the solutions generated by the *Corridor Method* can be measured in absolute terms “how far are they from the respective optimal solutions?” and/or in relative terms ”how do they perform in comparison with solutions generated by other methods?” Both are important.

For obvious reasons it is impossible to provide a general-purpose theoretically-based recipe for the evaluation of the performance of the Corridor Method. This must be done experimentally.

As indicated above, there is a significant experience with the performance of a simplified version of the method in the water resources area (Heidari et al., 1971; Chow et al., 1975) where it is was deployed to assist DP cope with the Curse of Dimensionality in the context of reservoir control problems (Yakowitz, 1982).

By the same token, there are many practical instances of the TSP precedence constraints similar to C_1and C_2, which are satisfied with respect to a nominal tour and m is much smaller than n. In such cases linear time DP algorithms can be devised to generate optimal solutions to the TSP (Balas, 1999; Balas and Simonetti, 2001). This suggests the deployment of the *Corridor Method* with a smaller value of m – to speed up the algorithm – at the expense of not being able to guarantee the optimality of the solutions generated this way.

8. Discussion

Since its birth in the 1950s DP has been a prime target for the Curse of Dimensionality. It is not surprising therefore that over the years ways have been found to speed up the performance of DP algorithms and to use DP to generate

”good” rather than optimal solutions. Indeed, the later predates the birth of classical metaheuristics.

In any case, it is common practice to deploy DP in conjunction with other methods such as BB (Morin and Marsten, 1976; Ibaraki, 1987; Carraway and Morin, 1988) and Composite Concave programming (Sniedovich, 1992) to deal with the computational aspects of DP.

In our discussion we have shown that the basic idea behind the heuristic method DDDP that was originally developed to support DP can be generalized and framed as a modern metaheuristic — more specifically as an iterated local search method — characterized by the large, method-based neighborhoods that it deploys. It should be stressed that DDDP is just one of many other similar schemes — often called iterative DP — where DP is deployed iteratively to improve an incumbent solution (Kossmann and Stocker, 2004; Luus, 2000; Leung et al., 2004)

As was illustrated in Example 4.3, DP-based neighborhoods may also be suitable for deployment by other optimization methods. This is not surprising given, for example, the polyhedral characterization of discrete DP models (Martin et al., 1990) and the close relationship between DP and BB (Ibaraki, 1987). Hence, both conceptually and technically the *Corridor Method* paradigm is readily applicable in the context of a vast class of combinatorial optimization problems and can support a number of major implicit enumeration methods.

This applies to metaheuristics as well: the *Corridor Method* can be used as a tool for improving the solution generated by other metaheuristics, which can be fed as a seed to the Corridor Method. By the same token, the *Corridor Method* can be used to generate seeds for other metaheuristics.

It should be noted that as is the case of most other metaheuristics, a number of important implementation issues must be dealt with on a case by case basis. In particular,

- What rules should be used to generate new seeds?
- What should be the width of the corridor?
- How “good” is a solution generated by the method?

And of course there is always the issue of competitiveness, especially in relation to other metaheuristics.

9. Conclusions

We have shown that the *Corridor Method* provides an intuitive and constructive framework for the formulation of metaheuristics based on local search over large neighborhoods via the imposition of exogenous constraints on the decision space of the target problem. This framework can be applied in a systematic and unified manner to extend the scope of operation of existing optimization methods. It also sheds new light on the interplay between neighborhood definitions and local search based metaheuristics.

Although the method has a very strong DP flavor, we have shown that it can be applied to other major optimization methods such as LP, BB and even other metaheuristics. It would therefore be interesting to investigate how effectively it will perform in such environments.

The main objective of the discussion was to formalize the method as a metaheuristic and to relate it to other metaheuristics. This done, the natural next step would then be to test the method against other metaheuristics

Acknowledgment

The authors wish to thank Art Lew and Seiichi Iwamoto for their constructive comments on the paper.

References

Ahuja, R.K., Ergun, O., Orlin, J.B., and Punnen, A.P. (2002) A survey of large-scale neighborhood search techniques. *Discrete Applied Mathematics* **123**(1-3), 75-102.

Balas, E. (1999) New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research* **86**, 529-558.

Balas, E. and Simonetti, N. (2001) Linear time dynamic programming algorithms for new classes of restricted TSP's: a computational study. *INFORMS Journal on Computing* **13**(1), 56-75.

Bellman, R.E. (1957) *Dynamic Programming*. Princeton University Press.

Carraway, R.L., and Morin, T.L. (1988) Theory and applications of generalized dynamic programming: an overview. *International Journal of Computers and Mathematics with Applications* **16**, 779-788.

Chow, V.T., Maidement, D.M. and Tauxe, G.W. (1975) Computer time and memory for DP and DDDP in water resource systems analysis. *Water Resources Research* **11**(3), 621-628.

Dantzig, G., Fulkerson, D. and Johnson, S. (1954) Solution of a Large Scale Traveling Salesman Problem. *Operations Research* **2**, 393-410.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H. Freeman.

Heidari, M., Chow, T., and Kokotovic, P.V. (1971) Discrete differential dynamic programming approach to water resources systems optimization. *Water Resources Research* **7**(2), 273-282.

Hoos, H.H. and Stützle, T. (2004) *Stochastic Local Search*. Elsevier.

Ibaraki, T. (1987) *Enumerative Approaches to Combinatorial Optimization*. J.C. Baltzer AG, Basel.

Kossmann, D. and Stocker, K. (2000) Iterative dynamic programming: a new class of query optimization algorithms. *ACM Transactions on Database Systems* **25**(1), 43-82.

Leung C., Appleton, B and Sun, C. (2004) Fast Stereo Matching by Iterated Dynamic Programming and Quadtree Subregioning. In: A. Hoppe, S. Barman, and T. Ellis, eds. *British Machine Vision Conference* **1**, 97-106, Kingston University, London.

Luus, R. (2000) *Iterative Dynamic Programming.* Chapman and Hall, London.

Morin, T.L. and Marsten, R.E. (1976) Branch and bound strategies for dynamic programming. *Operations Research* **24**, 611-627.

Martin, R.K., Rardin, R.L. and Campbell, B.A. (1990) Polyhedral characterization of discrete dynamic programming. *Operations Research* **38**(1), 127-138.

Miller, C., Tucker, A. and Zemlin, R. (1960) Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM* **7**, 326 -329.

Potts, C. and van de Velde, S. (1995) Dynasearch - iterative local improvement by dynamic programming. Technical Report, University of Twente.

Sherali, H.D. and Driscoll, P.J. (2002) On tightening the relaxations of Miller-Tucker-Zemlin formulations for asymmetric traveling salesman problems. *Operations Research* **50**(4), 656-669.

Sniedovich, M. (1992) *Dynamic Programming.* Marcel Dekker, NY.

Voss, S. (2001) Meta-heuristics: The state of the art. In: Local Search for Planning and Scheduling, 1-23, LNAI 2148, Springer-Verlag.

Winston, W.L. (1994) *Operations Research: Applications and Algorithms*, 3rd edition. Thompson International, Belmont.

Yakowitz, S.J. (1982) Dynamic programming applications in water resources, *Water Resources Research*, **18**, 673-696.