The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs

Cees Duin,[1] Stefan Voß[2]

[1] University of Amsterdam, FEE, Institute A&E, Operations Research Group, Roeterstraat 11, 1018 WB Amsterdam, The Netherlands

[2] Technische Universität Braunschweig, Abt. Allg. BWL, Wirtschaftsinformatik und Informationsmanagement, Abt-Jerusalem-Straße 7, D-38106 Braunschweig, Germany

Received 7 November 1996; accepted 7 January 1999

Abstract: As a metaheuristic to obtain solutions of enhanced quality, we formulate the so-called pilot method. It is a tempered greedy method that is to avoid the greedy trap by looking ahead for each possible choice (memorizing the best result). Repeatedly, a so-called master solution is modified, each time in a minimal fashion to account for the "best" choice, where all choices have been judged by means of a separate heuristic result, the "pilot" solution. We apply the method to the well-known Steiner problem in a weighted graph, that is, the problem is to determine a subgraph of minimum total weight spanning a set of given vertices. The pilot method may be seen as a system for heuristic repetition. As a higher time complexity order is usually associated with repetition, we propose policies to reduce the running times, while retaining an enhanced solution quality. Where possible, to encourage application of the pilot method to other combinatorial problems, we formulate in general terms. © 1999 John Wiley & Sons, Inc. Networks 34: 181–191, 1999

1. INTRODUCTION

What can one do when existing exact methods are too slow and existing heuristics too inaccurate? As expected, there will be a trade-off between computation time and accuracy. To enable such a trade-off, without much effort, this paper proposes a strategy of heuristic repetition: a metaheuristic, called the pilot method. For the Steiner problem graphs (SPG), we formulate different applications as well policies to diminish the required computation time. During the discussion, we can reinterpret well-known heuristics for the SPG, for example, Rayward-Smith and Clare [16], Minoux [13], and Zelisovski [19].

For combinatorial optimization problems that are NP-hard, there are usually known algorithms that construct approximate solutions in polynomial time by means of quick greedy rules. As greedy choices are often myopic, these heuristics can fall into the so-called greedy trap, returning reasonable but not near-optimal solutions. In this respect, it seems important to look ahead, that is, to take somehow into account how present choices affect later

Correspondence to: C. Duin; e-mail: ceesd@fee.uva.nl

CCC 0028-3045/99/030181-11

choices in the algorithm. Traditionally, this is done in one pass of a proceeding that uses more sophisticated priority rules than greedy rules. To enhance the solution quality of an existing heuristic, the so-called pilot method is a practical method, requiring little extra algorithmic research and implementation. The basic idea is a look-ahead strategy that exploits multiple passes of the existing heuristic while changing the input variables.

A similar type of look-ahead strategies has already been extensively studied within the area of artificial intelligence, especially when dealing with models for game-playing programs (like chess among others). An excellent survey of this field was provided by Pearl [14]. Also, here, the idea is to examine all possible decisions with respect to their likelihood to yield a future advantage, to prune away unpromising decisions, and to choose decisions that are most promising. Within the area of combinatorial optimization problems, these techniques have not yet been fully exploited.

The pilot method is a tempered greedy method, based on the repetition of another heuristic, the so-called pilot heuristic (or subheuristic). As such, the method is a metaheuristic; it can be formulated to suit all kinds of combinatorial optimization problems. In Section 2, we give a general description ending with an illustration on a well-known prototype problem for combinatorial optimization: the Traveling Salesman Problem (TSP). More developed applications are presented in the third section for the Steiner problem in graphs (SPG). An instance of the TSP and the SPG is given by a weighted undirected graph $G = (V, E, c)$, where $V = \{1, \ldots, n\}$ is the set of vertices, E is the set of undirected edges, and c is a nonnegative weight function on E. The TSP requires to order all vertices in a minimum cost tour $(i_1, i_2), (i_2, i_3), \ldots, (i_n, i_1)$ of n edges. Also, the SPG takes as input a weighted undirected graph $G = (V, E, c)$. Additionally, there is given a subset of "basic" or "special" vertices, say $K \subset V$. The problem requires a connected subgraph T of minimum total edge weight, such that the vertex set of T includes K.

With the SPG as a vehicle of demonstration, we discuss the different strategies that can be adopted. Section 3.1 briefly reviews the SPG, in particular, the heuristics, that are to be used later as subheuristics. In Section 3.2, we consider different methods of heuristic repetition. Due to the heuristic repetition, it is expected that the pilot method entails a high time complexity. After describing several pilot methods, we discuss in Section 3.3, in both general and specific terms, policies to reduce the running time while preserving as much as possible the solution quality. In Section 3.4, we present computational results for the application of the methods described, using test problem beds from Duin and Voß [5]. The latter paper presents fast heuristics already attaining a good solution quality. It appears that one can substantially improve the solution quality, if these heuristics are run in the setting of a pilot method. Section 4 contains conclusions and perspectives on further research.

2. THE PILOT METHOD

Consider a combinatorial optimization problem defined on a finite set of elements E weighted by a cost function $c:E \to \mathbb{R}$. The problem is to select at minimum cost a subset (or solution) $S^* \subset E$, satisfying some feasibility properties. Let there be known an efficient heuristic H producing a solution which is not always satisfactory (in the sense that the objective value of the solution can deviate too much from the optimal objective value).

Imperatives for almost any heuristic are: be greedy, perform trial and error, fix variables, and look ahead. In the pilot method, these aspects of heuristic strategies come together in a simple setting.

By looking ahead with H as a "pilot heuristic" (or "subheuristic"), one is to build up cautiously a partial solution M, "the master solution." Separately for each element e not in M, the subheuristic is to extend tentatively a copy of M to a (fully grown) solution in such a way that e is included. Let $p(e)$ denote the objective value of the solution obtained by pilot H for $e \in E\backslash M$, and let e_0 be the most promising element according to the subheuristic, that is, $p(e_0) \leq p(e)$ for all $e \in M$. Element e_0 is included in the master solution M, by changing it in a minimal fashion. On the basis of the changed master solution M, new pilot calculations are started for each $e \notin M$, providing a new solution element e'_0, and so on. This process could proceed, for example, until further pilot calculations do not lead to improvement.

Alternatively, we can state the pilot method for combinatorial optimization by drawing a parallel with the classical branch-and-bound approach. Imagine a branch-and-bound algorithm for a combinatorial minimization problem that fully branches a (sub)problem to all possible values of a solution variable that is going to be fixed. As an algorithm, the pilot method is very much identical, applying the very same rules—visiting the nonfathomed subproblems in order of the best bound and fathoming a subproblem if the bound is worse than an incumbent—with only one essential difference: The heuristic pilot method bounds each (sub)problem with an *upper* bounding procedure, whereas the exact branch-and-bound algorithm would calculate a *lower* bound. As a better upper bound is of lower objective value, this leads to an algorithm that persistently chooses as the next subproblem to be branched the subproblem of the new incumbent solution. This is done as long as the incumbent moves further down, to a child problem with a better upper bound. If not, one stops, returning the incumbent as final solution (all open subproblems are "fathomed" with a bound equal or worse than the incumbent).

Suppose that the applied upper bound procedure, that is, the subheuristic, takes time $O(n^k)$, where n represents the

size of the problem instance and k is a positive integer. Then, the pilot method takes time $O(n^{k+2})$, assuming that there are $O(n)$ elements in a feasible solution. Although the depth of the branch-and-bound tree is $O(n)$ and each branching gives at most $O(n)$ new subproblems, there is each time only one branch that leads to a most promising "not-fathomed" child problem.

We now illustrate the method on the TSP; see, for example, Lawler et al. [11]. The nearest-neighbor heuristic will be used as a subheuristic. Let us first describe this heuristic: Starting with vertex $i_1 = 1$, one chooses greedily the edge that leads to a nearest node i_2 of i_1, then from i_2 one adds to the partial solution $\{(1, i_2)\}$, the edge to a vertex $i_3 \in V\backslash\{i_1, i_2\}$ that is nearest to i_2, and so on, until (i_{n-1}, i_n) has entered the solution; then, (i_n, i_1) concludes the tour.

Imagine a branch-and-bound algorithm for the TSP that successively fixes edge variables $x_1, x_2, \ldots, x_n$ of the tour; for edge x_1 starting with vertex 1, we must fix the other incident vertex; then, in a child problem, the other vertex of edge x_2 adjacent to x_1 is to be fixed and so on. With the nearest-neighbor heuristic as upper-bounding procedure, we first produce the ordinary heuristic result on the main problem, say of objective value $p(1)$. Then, the nearest-neighbor heuristic is separately performed on $n - 1$ subproblems, each time with a different first edge $x_1 = (1, i)$ being fixed, thus producing pilot results $p(i)$ for $i = 2, 3, \ldots, n$. The subproblem with the best bound, say $x_1 = (1, i_0)$ has pilot result $p(i_0) = \min_{i \in V\backslash\{1\}} p(i)$, is chosen for further branching [effectively, edge $(1, i_0)$ is permanently accepted as first edge of the partial solution]. The bounds on the $n - 2$ subproblems, one for each $x_2 = (i_0, j)$, with $j \in V\backslash\{1, i_0\}$, are obtained by running the nearest-neighbor heuristic from the fixed partial solution $\{(1, i_0), x_2\}$. The subproblem, say $x_2 = (i_0, j_0)$, with the best pilot result is further branched, that is, a second edge permanently enters the partial solution. The process continues until all variables are fixed.

The quickest stopping rule would be to stop as soon as none of the child nodes has a better pilot result than the parent node. In this example, we favor the less quicker stopping rule that continues the branching also when the best pilot result matches the former incumbent result. The reason is that branching in a later stadium can still lead to a better solution. This may be also true for other nonchosen subproblems having a bound equal to the incumbent (or almost equal)—so why not branch them also (or allow for hill climbing). The answer is simply that we cannot do this in the context of a *heuristic* of polynomial time complexity. On the other hand, the strength of the method is that it already applies much trial and error and hill climbing. When in a subproblem the next edge is fixed, we do perform hill climbing if this edge is unattractive. The pilot heuristic is there to check whether a mature solution with this edge can still be most attractive; if so, it will be chosen. Thus, the pilot method is a tempered greedy method.

To get an enhanced solution by means of the pilot method, a necessary condition for the subheuristic is that it can return a better objective value when one has fixed one or more elements into the solution. Most greedy heuristics satisfy this precondition, but in the end, numerical experiments should clarify if and to what extent the associated pilot method really does enhance the stand-alone result.

3. APPLICATION TO STEINER'S PROBLEM IN GRAPHS

Steiner's problem in graphs considers an undirected weighted graph $G = (V, E, c)$ with a weight c_{ij} on each edge $(i, j) \in E$ and with in V, the set of vertices, a given subset of "special" vertices K. The problem is to find a connected subgraph T of minimum weight, $c(T) = \sum_{(i,j)\in E(T)} c_{ij}$, such that $V(T)$ includes K. We can assume nonnegative edge weights; then, at least one optimal solution is a tree, a so-called Steiner tree.

The SPG and important special cases thereof are NP-hard; see Garey and Johnson [7]. However, it also comprises two basic problems that are well solvable: These are "the shortest path problem" for $|K| = 2$ and "the minimum spanning tree (MST) problem" for $K = V$. Generally, the optimal solution is an MST, but not necessarily on (the subgraph induced by) K. Solutions of lower cost can be obtained by spanning some set $K \cup W$, where W is a set of so-called Steiner vertices from $V\backslash K$. A comprehensive book on the Steiner problem is that of Hwang et al. [8].

In section 3.1, we discuss Steiner tree heuristics that either relate to the pilot method or are exploited later as subheuristics. For a more comprehensive survey, the reader is referred to Duin and Voß [5]. In Section 3.2, we discuss several options for repetition. In the next section, increasingly accurate pilot methods for the SPG are designed, together with a discussion of speed-up policies. More than once we will reinterpret existing heuristics within the pilot setting, providing a different view on these methods. Finally, extensive numerical results are given showing the impact of a pilot setting.

We will make use of the following terminology and notation:

- For a subgraph H of $G = (V, E)$, the set **V(H)** contains the vertices of H, and **E(H)** contains the edges of H, but when clear from the context, *we may identify subgraph H with its set of vertices or its set of edges,* for example, writing $k \in H$ instead of $k \in V(H)$ or $(i, j) \in H$ instead of $(i, j) \in E(H)$.
- For a finite set H and a real-valued function b: $H \to \mathbb{R}$, we write $b(H)$ for the sum $\sum_{h\in H} b(h)$. For instance, for a tree T in G, $c(T)$ denotes the total weight of the edges in T.

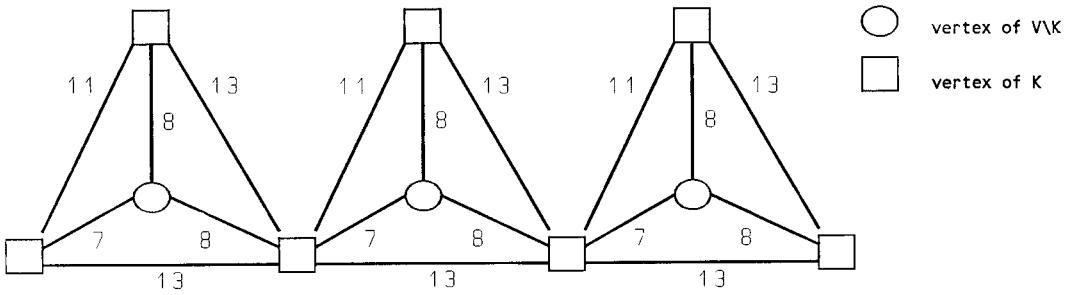

Fig. 1. PH finds all optimal Steiner vertices, whereas TH-V finds only one.

- Given a tree T spanning K, the so-called crucial set $\mathbf{V_c(T)}$ is defined as $V_c(T) = K \cup \{i \in V(T)\backslash K : i \text{ has degree } \geq 3\}$.
- The distance graph **(A, d)** for a subset $A \subset V$ is the complete graph with vertex set A, and for all $a, b \in A$ edge weights $\mathbf{d_{ab}}$, the shortest path distance from a to b in $G = (V, E, c)$ with respect to c.
- For any tree T spanning a set of vertices V and a subset $A \subset V$, we write $\mathbf{T_A}$ for the minimal subtree in T that spans A.

3.1. Steiner Tree Heuristics

The first successful heuristics for the SPG build up a *partial solution* by the insertion of shortest paths. The *partial solution T* consists of a number of component trees that are locally near optimal. Initially, there are $|K|$ component trees $T_k = (\{k\}, \varnothing)$, one for each special vertex $k \in K$, and in $O(|K|)$ steps, the number of components reduces to one, as T grows into one feasible solution. As the final solution T need not be a minimum spanning tree on $V(T)$, it has become common practice to conclude with the posterior procedure "**MSTPRUNE**":

Determine MST T' on $(V(T), c)$; T'_K the minimal subtree of T' spanning K can be a better solution. Originally, MSTPRUNE was the concluding step of the heuristic of Kou et al. [10]; the first two steps are known as the **distance graph heuristic:**

1. Determine on the distance graph (K, d) a spanning tree S of minimum weight $d(S)$;
2. Form tree T in G by replacing each "d-edge" of S for (the necessary part of) a shortest path.

An implementation of the distance graph heuristic running within $O(|V|^2)$ time is due to Mehlhorn [12].

The algorithm of Takahashi and Matsuyama [17], for short **TH**, resembles Prim's algorithm [15] for finding an MST. All components in the partial solution remain a singleton, except one: the growing component T_{k0}, where $k_0 \in K$ is a "start vertex." In $|K| - 1$ steps, the nearest other "component" (here a special vertex) is selected and connected to T_{k0} by means of a shortest path, that is, TH can be seen as a nearest neighbor method (or as cheapest insertion method). By putting each time zero weights on the edges that enter T_{k0}, one can implement TH with $|K| - 1$ shortest path computations, so the computational complexity is $O(|K||V|^2)$.

The heuristic of Rayward-Smith and Clare [16], for short **RH**, also unites repeatedly two components of a growing partial solution T. The vertex v_0, that minimizes a heuristic function R: $V \to \mathbb{R}$, is incorporated into T by insertion of shortest paths from v_0 to T_0 and T_1, the nearest and the second-nearest component to v_0. Results were presented for an $O(|V|^3)$ heuristic based on a function R that represents the minimum average distance to at least two of the components left in T (say there are $\sigma + 1 \leq |K|$ components left):

$$R(v) := \min\left\{\frac{1}{t}\sum_{i=0}^{t} d(v, T_i) \mid t = 1, \ldots, \sigma\right\},$$

where $d(v, T_i) = \min\{d_{vt} | t \in T_i\}$ denotes the distance to T_i, in order $d(v, T_i) \leq d(v, T_{i+1})$ for $i = 0, \ldots, \sigma$.

Minoux's heuristic [13], briefly **MH**, exploits the sensitivity in MST S on (K, d) to select Steiner vertices:

1. For $v \in V\backslash K$: $\text{win}(v) = d(S) - d(S_v)$, S_v being an MST on $(V(S) \cup \{v\}, d)$;
2. If $\text{win}(v_0) = \max_v \text{win}(v) > 0$, then $S := S_{v_0}$, return to *step 1;*
3. Replace the distance graph edges of S by associated shortest paths to obtain a solution in G.

With a linear time subroutine to update an MST for an extra vertex, MH has an $O(|V|^3)$ time bound.

The heuristic "**SVERTEX**" of Duin and Voß [5] produces a solution quality comparable to that of MH. Further, as an alternative for TH, they formulated the heuristic "**RECONSTRUCT.**" Both algorithms take as input all-pair shortest paths and a feasible solution S, for example, S as an MST on (K, d). In a number of steps, S is transformed into a solution of possibly lower cost, by means of a subroutine for path and/or vertex exchange; more details are given in

Section 3.3. The body of algorithm reConstruct (without the initialization that computes all-pair shortest paths) has an $O(|K||V|)$ running time. With a higher coefficient, this is also true for Svertex on most problem types; however, to guarantee it in the worst case, one should install limits to bound the number of exchanges in the subroutine; see Duin and Voß [5].

3.2. Enhancing the Solution Quality by Repetition

For a Steiner tree T, it is not mandatory to communicate all the Steiner vertices of T. Only the crucial set $V_c(T)$ is of interest; remember that $V_c(T) = K \cup \{i \in V(T) \backslash K \mid i \text{ has degree } \geq 3\}$. When well-shaped, T consists of shortest paths between the crucial vertices; when shaped optimally with respect to $V_c(T)$, T represents an MST on $(V_c(T), d)$. To improve the solution T, one should change the crucial set; this may happen in the following approach:

MultiplePass
input: a solution T for (V, K, E, c), say by heuristic Xh
output: a solution T_0
repeat
$K' := V_c(T)$;
$S :=$ new heuristic solution for (V, K', E, c)
(* *for new set of special vertices* K' *);
$T_0 := T$; $T := S_K$;
(* *pruning off pseudo special vertices of* $K' \backslash K$ *)
until $c(T) \geq c(T_0)$;

One could determine S with the same heuristic Xh, which obtained the input solution T. In Duin [3], this easy MultiplePass method was tested successfully for the heuristics Th and Rh, giving improvements in up to 50% of the test cases, whereas these heuristics already included MSTprune.

Note that MultiplePass will not be effective for Mh. On the other hand, Mh already is a MultiplePass method, when using an optimal 1-Steiner tree solution for K' in (V, d) as a heuristic solution for (V, K', E, c). (The polynomially solvable 1-Steiner tree problem restricts the number of Steiner vertices to at most one.)

In various combinatorial contexts, a method like MultiplePass might be helpful, for example, if the answer to the following question is affirmative: When providing some elements of a (near) optimal solution, can this information let a heuristic run better? However, such circumstances can call for a pilot method to be applied, too.

Conspicuous about Th is that the result depends on the choice of the start vertex. One could choose it out of K, trying a nonspecial vertex as the Steiner vertex. The heuristic Th-V (see Winter and MacGregor Smith [18]), runs Th for all possible starts $v \in V$ and picks out a best solution. So, Th-V also enhances the solution quality of a heuristic by using multiple passes; being a trial-and-error method, it is quite different from MultiplePass described above. The pilot method on the SPG, as first applied with Th as a pilot heuristic, starts similarly as Th-V, but continues. At any time, the so-called master solution, a partial solution T, is a forest of trees: $T_1, T_2, \ldots, T_\sigma$, covering the vertices of K—initially: $T = \bigcup_{k \in K} (\{k\}, \emptyset)$. In copies of this graph, with the components conceived of as special vertices (as if merged into a single vertex), one runs, separately for each vertex v, the heuristic Th with start v. Let vertex v_0 produce the best objective value. Then, two components of master solution T are permanently joined in a rerun of Th from v_0, this time in the original graph for two iterations (or one if v_0 is special). Repetition of this process constitutes a pilot method for the SPG, denoted here as Ph.

Alternatively, one can interpret Ph as a synthesis of Rh and Th: The formula $R(v)$ is replaced by $p(v)$, the objective value being returned by subroutine Th with start v. The heuristic Ph can produce better solutions than can Th-V, because of the fact that it can find more than one (local) improvement; see Figure 1. As explained later in Section 3.3, one can implement Ph in $O(|V|^3 + |K|^2|V|^2)$ time.

Table I shows the results of Ph in comparison with its forerunners on four test problem beds. A complete description of the test problems can be found in Duin and Voß [5]. For a given number $|V| = v$, each line of Table I gives the average and the maximum gap with respect to the optimal objective value (or when bracketed a lower bound) over a set of 100 instances (45 in the rectilinear case) with different densities of $|K|$ and or $|E|$. All the heuristics were concluded

TABLE I. Results of four related heuristics

Size	Th		Rh		Ph		Th-V	
v	a%	m%	a%	m%	a%	m%	a%	m%
Random weights								
80	1.4	10.3	0.9	9.8	0.1	1.9	0.2	2.6
160	1.4	14.2	0.7	4.7	0.1	2.2	0.4	4.1
320	2.3	9.3	1.2	7.0	0.3	3.7	0.8	3.8
Euclidean weights								
80	1.7	8.4	0.9	7.9	0.1	4.2	0.5	4.2
160	2.3	11.0	1.0	5.0	0.2	2.9	0.7	4.1
320	2.4	7.9	1.3	7.9	0.2	2.3	1.0	4.4
Incidence weights								
80	14.4	41.0	1.7	9.3	0.9	6.5	1.6	6.5
160	17.2	43.7	1.8	10.5	1.2	5.0	2.2	7.6
320	(19.2)	45.5	(1.9)	(6.7)	(1.5)	(7.6)	(2.5)	(10.2)
Rectilinear (45 per line)								
81	3.4	12.8	1.2	9.1	0.1	1.6	0.7	4.3
169	4.1	11.8	1.4	7.1	0.3	2.5	1.2	5.6
324	5.0	10.2	1.8	6.3	0.3	1.9	1.7	4.5

m%, a%: Over 100 instances per line (over 45 instances when thus indicated), the maximum and average gap in percent of optimal value (or a lower bound, when bracketed).

with the improvement procedure MSTPRUNE. The optimal values were computed with the branch-and-bound program of Duin [3]; within a limit of 2000 subproblems, nine instances, all of incidence type, were not solved to optimality.

In relation to its forerunners RH and TH-V, the heuristic PH returns superior solutions, but there is the drawback of a higher time complexity. However, with policies as formulated in Section 3.3, one can decrease the computational needs of PH to a level comparable to that of RH and TH-V, without giving up its superiority in terms of attained solution quality.

In fact, we have already reviewed a pilot method for the SPG. Not only can one interpret heuristic MH as a MULTIPLEPASS method, it also is possible to interpret MH as a pilot method. Remember that the SPG alternatively requires an optimal set of nonspecial crucial Steiner vertices. Now, imagine a pilot method as a branch-and (upper)-bound algorithm on variables $x_1, x_2, \ldots$ (at most $|K| - 2$), where x_i is "the i-th" nonspecial crucial vertex in the Steiner tree. Branch at level i to all vertices left in $V\backslash K$ for variable x_i, and execute as the upper-bound procedure H0 (the first step of) the distance graph heuristic on $K \cup \{x_1, \ldots, x_i\}$. The resulting pilot method PH0 is equivalent to heuristic MH. Computational results for PH0 can be found in Table III, Section 3.4.

3.3. Tailoring a Pilot Method

For the SPG, we first design pilot methods based on more accurate pilot heuristics than TH. Second, we formulate in general terms shortcut policies to reduce the running time of a pilot method. Specific implementations of these policies are given for the SPG. The next section will show how the solution quality responds.

As more accurate subheuristics, we use RECONSTRUCT and SVERTEX of Duin and Voß [5]; we further review some aspects of these algorithms. The subroutine EXCHANGE_FOR(q), which both algorithms use for path and/or vertex exchange, processes a vertex q. It either rejects or accepts q as new crucial vertex in the candidate solution S; in the latter case, q enters S together with incident paths at the expense of other paths of S and possibly also former crucial vertices of S are then expelled by q. One of the differences between RECONSTRUCT and algorithm SVERTEX is that the latter has exploited this subroutine exhaustively before changing S and this leads to a better average solution quality at the expense of computation time.

When using RECONSTRUCT as a pilot, the aspect of vertex exchange invites us to loosen the system of a pilot method. Previously, PH and MH *extended*, on the basis of pilot results, a master solution being a *partial* solution and a *partial* crucial set, respectively. Now, we just say that we are to *modify* a master solution on the basis of pilot results to a solution of lower cost (with probably more Steiner vertices).

We formulated a pilot method on the basis of RECONSTRUCT in Figure 2. To run RECONSTRUCT on a vertex $v \notin S$, we first include it in S, by forming the distance MST on the crucial vertex set of S plus vertex v. Step (1) evaluates all vertices that are presently not in the crucial set. Hill climbing is possible, because after step (1a), the objective value of S_v can be higher than $d(M)$ and still decrease below $d(M)$ after transforming S_v in RECONSTRUCT, that is, applying EXCHANGE_FOR(q) for all path vertices q of S_v, including also the implicit (noncrucial) vertices. For a vertex v_0, argument of the best pilot solution, the master solution M is to follow a minimal part of the transformation steps to S_{v0} by RECONSTRUCT. We do not even enter this procedure if acceptance of v_0 in the MST gives us already a lower cost; otherwise, RECONSTRUCT is exited as soon as after a transformation step $d(M_0) < d(M)$. This reflects the idea of changing the master solution to a more mature solution in a minimal fashion. In the sequel of this section, we discuss methods to speed up a pilot method.

In each iteration of PH1, the work load is full: $O(|V|)$ times the operations for RECONSTRUCT, summing up to a rather high time complexity. To diminish the time requirements, one can resort to parallel processing, obtaining different pilot solutions simultaneously. However, there are also other prospects to speed up a pilot method:

(i) Another implementation than the straightforward one may reduce the time complexity.
(ii) Limit the number of iterations by modifying the master solution each time to a greater extent.
(iii) Obtain pilot values in iteration *iter* for a limited number of vertices, say *pilots*(*iter*).
(iv) Apply short cuts in the calculation of pilot values, that is, approximate them.

Naturally, ideas with regard to (i) are most welcome, as implementing them cannot worsen performance. By efficiently updating solutions, algorithm MH, when interpreted as the pilot method PH0, has, in fact, followed this approach. When branching to all values of x_{i+1}, each new distance graph solution can be obtained from the incumbent, the distance MST on $K \cup \{x_1, \ldots x_{i-1}\}$ in linear time by means of an MST update procedure.

Let us now apply approach (i) on PH1. Because of the initialization, that is, calculating all shortest paths, RECONSTRUCT would require $O(|V|^3)$ time as a stand-alone heuristic, but the procedure itself, transforming S, requires $O(|K||V|)$ time. In the implementation of PH1, the shortest path determination is a one-time task, so the pilot method of Figure 2 requires $O(iter\ |K||V|^2)$ time, which is more moderate than a straightforward implementation of $O(iter\ |V|^4)$ time. Similarly, the pilot method PH can take $O(|V|^3$

```
PH1
input:  (V,K,d,r), where d is a |V|x|V| matrix of shortest distances in (V,E,c) and r is a matrix for
        the retrieval of associated shortest paths (for path/vertex exchange in subroutine RECONSTRUCT)
output: M a solution for the SPG on (V,K,E,c);
BEGIN
iter := 0;   M := MST of (K, d);
repeat
iter := iter+1;
(1) for v ∉ M do                                   (*test each vertex not in the present crucial set*)
    (a) S_v := MST of (V_c(M) ∪ {v}, d);           (*initialize v's pilot solution*)
    (b) RECONSTRUCT(S_v);                          (*RECONSTRUCT transforms S_v to a pilot solution*)
    (c) p(v) := d(S_v);
(2) v_0 := arg min p(v);                           (*select a vertex with the minimum objective value*)
(3) if p(v_0) < d(M) then                          (*modify the master solution to include v_0*)
    (a) M_0 := MST of (V_c(M) ∪ {v_0}, d);
    (b) if d(M_0) ≥ d(M) then
            RECONSTRUCT(M_0), but exit this subroutine as soon as: d(M_0) < d(M);
    (c) M := M_0
until p(v_0) ≥ d(M);
Substitute the 'd-edges' of M by shortest paths;
END
```

Fig. 2. A pilot method for the SPG, based on pilot RECONSTRUCT.

$+ |K|^2|V|^2$) time, because the body of TH runs in $O(|K||V|)$ time, too (see Duin and Voß [5]), and in PH, there are at most $|K| - 1$ iterations.

When following approach (ii), (iii), and/or (iv), the average solution quality can deteriorate. Successful ideas with respect to these approaches are to obtain a reduced running time without increasing too much the possibility to get returned a worse solution. We first discuss approach (ii):

Most rigorously, one can fully transform the master solution M, that is, do not exit RECONSTRUCT(S_{v0}) prematurely. (By the way, such a complete transformation in pilot method PH would lead to the method MULTIPLEPASS on TH-V.) However, a complete transformation seems to be in conflict with the aspiration of the pilot method to avoid greedy pitfalls. A too thorough change of the master solution S can block the subheuristic in its ability to put forward new solution elements. If the master solution acquires many Steiner vertices right away, the path lengths in S become shorter, making a greedy replacement in the new pilot runs of RECONSTRUCT more difficult. A more moderate policy is GO(⅓): Exit the pilot modifying M_0, as soon as (at least) one-third of the potential number of transformation steps is executed.

As a first application of approach (iii), we suggest the policy DROP. It executes the pilot in iteration $iter = 1$ for all nonspecial vertices, $pilots(1) = |V \backslash K|$, but in iteration $i + 1$, we intend to run only $pilots(i + 1) = pilots(i)/2$ pilots (rounded-up integer). One-half of the previous vertex set is to drop off, those vertices that returned a pilot result worse than the median result in iteration i. That is, a drop policy lets former pilot results rule out unattractive vertices. Following this approach, there are at most $\log|V|$ changes of the master solution M with in total $O(|V|)$ executions of the subheuristic. Thus, if the drop strategy is adopted in Figure 2, the running time becomes $O(|K||V|^2)$.

Like MH, the algorithm of Zelikovsky [19] also modifies a solution on the basis of a pilot: $p(v) = d(S_v)$, with S_v being a 3-restricted MST of $(V(S) \cup \{v\}, d)$, that is, an MST that requires v to have degree 3. In fact, an improved running time can be obtained for this heuristic, due to a drop policy that is valid (that cannot deteriorate the final result): Rule out a vertex v from consideration in subsequent iterations if $p(v) = d(S_v) \geq d(S)$; see Duin and Voß [5].

The policy FILTER to implement approach (iii) is quite different. At the start of each new pilot iteration, none of the elements v is a priori ruled out. But each time, in a prepilot

TABLE II. Stand-alone results of H0 (distance graph MST), H1 (reConstruct), and H2 (Svertex)

Size	H0			H1			H2		
v	a%	m%	#opt	a%	m%	#opt	a%	m%	#opt
Random weights									
80	10.9	36.4	2	0.6	10.3	70	0.23	4.9	83
160	12.9	29.9	0	0.8	5.3	43	0.35	4.7	64
320	14.3	26.7	0	1.0	7.4	33	0.56	6.2	51
Euclidean weights									
80	8.8	20.8	7	1.1	7.9	46	0.20	3.4	79
160	11.0	33.4	1	1.2	7.8	26	0.35	4.6	63
320	10.9	32.2	0	1.5	7.6	9	0.34	2.5	44
Incidence weights									
80	28.8	45.9	1	11.4	41.0	12	0.98	11.8	51
160	31.6	45.0	0	14.6	43.7	9	0.96	5.2	34
320	(35.2)	45.5	0	(17.1)	45.5	5	(1.0)	6.0	(33)
Rectilinear (45 per line)									
81	6.6	17.3	2	0.9	8.5	24	0.03	0.5	42
169	9.5	20.0	0	1.2	7.4	12	0.49	5.6	27
324	10.3	20.1	0	1.3	4.7	5	0.40	3.9	22

See footnote to Table I.

phase, one is to filter out from V a number of candidate vertices among which v_0 is to be determined. For all $v \in V$, a much quicker prepilot procedure is run with outcome $pp(v)$. In iteration i, the vertices of lowest outcome $pp(v)$ are candidates for a full pilot run, so one determines the computationally more expensive pilot value $p(v)$ for, say, $pilots(i)$ vertices only.

Our implementation of Filter on the SPG uses as a prepilot procedure for vertex v, a shortened version of reConstruct, one that processes in subroutine Exchange_for only the vertex v. Each time for a fixed number of $pilots(i) = 2|V|/|K|$ vertices, the full values $p(v)$ are determined. Like Drop, Filter also exhibits a running time of $O(|K||V|^2)$.

Encouraged by the better quality of PH1 when compared to PH, we experimented further, replacing in Figure 2 the heuristic reConstruct by the more accurate heuristic Svertex, thus obtaining a new pilot method called PH2. In the following sense, PH2 is an example of approach (iv): With Svertex as a pilot, we approximate a *second-order pilot method:* one that uses repeatedly the pilot method MH to calculate values $p(v)$. Both the method and the solution quality of Svertex is comparable to that of MH. Svertex can be seen as a faster approximated version running in $O(|V|^2)$ time.

Another more simple example of approach (iv) is to limit the running time of *each* pilot run: In our example, limit the number of transformation steps in reConstruct, for example, process at most four vertices in the subroutine for path/vertex exchange. We do not recommend this policy; it may reintroduce short-sightedness, as then pilot values do not represent (full) objective values (of fully grown solutions). Restricting the number of iterations in pilot runs is not in line with the intention of the pilot method (*P*erform *I*mproved *L*ook-ahead with *O*bjective-value *T*ests).

3.4. Computational Results

In this section, we give computational results for the pilot methods as well as the shortcut policies discussed in the previous section. For none of the listed algorithms have we run a posterior improvement procedure. We tested on the same four problem beds as used in Section 3.2. Table II shows how the heuristics that are to be used as subheuristics perform as standalone heuristics. The results are given in terms of average and maximum relative gaps with the optimal solution (or, when unavailable, with respect to a lower bound). Furthermore, we provide in column "#opt" the number of optimal solutions obtained in the sample of 100 or 45 problem instances. The results show that reConstruct and Svertex perform increasingly better, but their performance on the Incidence problems remains unsatisfactory.

Table III gives the results of three corresponding pilot methods: PH0, equivalent to MH, is a pilot method based on the distance graph heuristic; PH1 is given in Figure 2 of Section 3.3; and PH2 is analogous to the latter with Svertex replacing reConstruct. The results show that the use of a more accurate subheuristic indeed pays off in terms of solution quality. (The results between brackets on the Inci-

TABLE III. Performance of full pilot methods (P_H) based on, respectively, pilots H0, H1, and H2

Size	PH0			PH1			PH2		
v	a%	m%	#opt	a%	m%	#opt	a%	m%	#opt
Random weights									
80	0.18	4.9	86	0.00	0.0	100	0.00	0.0	100
160	0.36	4.7	64	0.00	0.2	98	0.00	0.0	100
320	0.60	6.2	50	0.01	0.5	95	0.01	0.5	99
Euclidean weights									
80	0.18	3.4	80	0.01	0.3	97	0.00	0.0	100
160	0.40	4.6	62	0.03	1.1	92	0.00	0.0	100
320	0.30	2.2	51	0.03	1.2	89	0.00	0.1	96
Incidence weights									
80	1.13	11.8	47	0.23	5.5	82	0.02	1.4	96
160	1.10	6.2	34	0.36	4.4	66	0.04	1.2	90
320	(1.1)	8.8	(33)	(.74)	(7.1)	(52)	(.26)	(4.1)	(82)
Rectilinear (45 per line)									
81	0.07	2.7	43	0.00	0.0	45	0.00	0.0	45
169	0.40	5.6	31	0.01	0.2	43	0.01	0.5	44
324	0.37	3.9	20	0.03	0.4	38	0.00	0.1	43

See footnote to Table I.

dence problems, especially those for PH2, are likely to be too pessimistic, as the measurement involves nine relatively poor lower bounds.)

In Tables IV and V, one can see how the methods PH1 and PH2 are affected by adopting one or more of the speed-up policies. Under the headings FILTER, DROP, and GO, we

TABLE IV. Performance of shortcut policies for PH1

Size	GO(⅓)			DROP			FILTER			RUSH		
v	a%	m%	t%	a%	m%	t%	a%	m%	t%	a%	m%	t%
Random weights												
80	0.00	0.0	56	0.01	0.8	35	0.04	1.9	25	0.09	2.1	11
160	0.01	0.4	39	0.01	0.5	20	0.02	0.4	12	0.16	2.2	3
320	0.01	0.5	29	0.07	2.2	11	0.08	2.6	6	0.18	2.0	1
Euclidean weights												
80	0.01	0.3	61	0.01	0.6	38	0.01	0.3	29	0.11	1.9	14
160	0.06	2.3	43	0.03	1.1	23	0.07	1.1	14	0.24	2.6	4
320	0.02	0.4	31	0.06	1.1	13	0.10	1.5	7	0.35	2.1	1
Incidence weights												
80	0.38	5.5	50	0.37	5.5	40	0.28	4.0	30	0.58	5.5	18
160	0.49	4.4	34	0.50	4.4	26	0.41	3.2	15	0.84	4.6	7
320	(0.84)	(7.1)	21	(.87)	(7.4)	13	(.72)	(6.5)	7	(1.3)	(8.4)	3
Rectilinear (45 per line)												
81	0.00	0.0	65	0.05	2.1	41	0.01	0.4	23	0.16	3.6	10
169	0.01	0.2	49	0.04	1.1	23	0.05	1.2	12	0.32	2.4	3
324	0.07	0.7	40	0.09	0.9	14	0.08	0.6	7	0.46	2.6	1

m%, a%: Over 100 (or 45) instances per line, the maximum and average gap in percent of optimal value (or a lower bound).

t%: Computation time as a percentage of the time used by the full pilot method PH1.

TABLE V. Performance of shortcut policies for PH2

Size	GO(⅓)			DROP			FILTER			RUSH		
v	a%	m%	t%	a%	m%	t%	a%	m%	t%	a%	m%	t%
Random weights												
80	0.00	0.0	57	0.00	0.0	40	0.00	0.4	15	0.03	0.8	5
160	0.00	0.0	40	0.00	0.1	25	0.01	0.4	9	0.07	1.8	2
320	0.01	0.5	29	0.01	0.5	16	0.06	0.7	5	0.08	0.7	1
Euclidean weights												
80	0.00	0.0	61	0.00	0.0	44	0.00	0.4	16	0.00	0.3	6
160	0.01	0.6	46	0.00	0.1	28	0.03	1.1	9	0.05	1.3	2
320	0.00	0.1	35	0.01	0.6	19	0.02	0.5	6	0.08	1.2	1
Incidence weights												
80	0.02	1.4	55	0.02	1.4	46	0.06	1.4	13	0.15	2.0	6
160	0.05	1.2	45	0.04	1.2	38	0.13	1.6	7	0.19	2.3	2
320	(0.28)	(4.2)	34	(.31)	(4.1)	30	(.36)	(4.3)	4	(.51)	(4.9)	1
Rectilinear (45 per line)												
81	0.00	0.0	67	0.00	0.0	47	0.01	0.4	15	0.00	0.2	6
169	0.01	0.5	48	0.01	0.5	29	0.05	1.3	9	0.06	1.3	2
324	0.00	0.2	39	0.01	0.1	21	0.02	0.6	6	0.12	1.8	1

m%, a%: Over 100 (or 45) instances per line, the maximum and average gap in percent of optimal value (or a lower bound).

t%: Computation time as a percentage of the time used by the full pilot method PH2.

tabulate the results under these policies. Their application to the SPG is described in the previous section; as FILTER policy for PH2, we used the same procedure as for PH1. Besides the pure policies, we also considered a policy named "RUSH": It applies FILTER, DROP, and GO simultaneously. (More precisely, in the first iteration, the prepilot procedure is run on $|V| - |K|$ vertices to select a set of $2|V|/|K|$ vertices for full pilot evaluation; in the next iteration, the prepilot runs only on the previously determined set to select $|V|/|K|$ vertices for a full pilot evaluation, etc. Meanwhile, the master solution is each time changed according to policy GO). Instead of column "#opt," we provide in Table IV the column "%t," giving the time requirements of the algorithm as a percentage of the time used by the associated full pilot method. The results show that one can control to a large extent the running time of the pilot method without too big a risk of spoiling the final solution.

4. CONCLUSIONS

We have formulated a method of heuristic repetition, the pilot method, to obtain an enhanced heuristic result. Successful applications to the Steiner problem in graphs were given, and with other such applications, we reinterpreted heuristics known from literature.

Straightforwardly, the pilot method leads to a high-order time complexity. However, we demonstrated that there are opportunities for a less time-consuming implementation by parallel processing, by an update of pilot solutions, or by other means. Moreover, to reduce time requirements, one can resort to shortcut policies with a slight tradeoff to the average solution quality. We described four such policies: Policy FILTER uses a prepilot procedure (quicker than the pilot procedure) to filter out only promising elements for the computationally more expensive full pilot examination. After each change of the master solution, the DROP policy runs the subheuristic for only a fixed fraction of the previously examined set of elements. Policy GO is aimed at having fewer iterations by changing the master solution more rigorously. Finally, an approximate but faster run of the subheuristic might turn out satisfactory.

Metaheuristic strategies such as simulated annealing, genetic algorithms, or tabu search appeal to the imagination with paradigms from, respectively, physics, biology, and historical bookkeeping. As a search method, the pilot method strides directly forward, never looking back but always looking fully forward, before each new choice. One can compare the pilot method with the system of a chess player: Before deciding on the next move, any chess player would very much like to have played a full separate game for each possible move. In real practice, a chess master must know his shortcut policies, filtering out only promising moves for a quick evaluation, running through the selected "pilot games" only as deep as it seems necessary.

For the Steiner problem in graphs, we have seen that the

pilot method can compete fully with other modern heuristic search techniques, for example, the simulated annealing approach of Dowsland [2], the genetic approaches of Kapsalis et al. [9] and of Esbensen [6], and the tabu search method reviewed in Duin and Voß [4]. On our extensive beds of test problems, a full pilot always delivered a good solution, very often optimal, the more so, when using a more accurate subheuristic. With a prudent speed-up policy, the method can couple a good solution quality with speediness.

Before embracing for a combinatorial optimization problem, a complicated search system using some sort of "artificial" intelligence, and suggesting the validity of this new system, one should in our opinion have surpassed with it the solution quality that can be obtained by a pilot method. From the theoretical point of view, we would like to have resolved the following: Are there (if possible, general, i.e., types of) combinatorial optimization problems and heuristics, such that an associated pilot method has a better worst-case error ratio than the pilot heuristic as a standalone heuristic?

REFERENCES

[1] E.W. Dijkstra, A note on two problems in connexion with graphs, Num Math 1 (1959), 269–271.

[2] K.A. Dowsland, Hill-climbing, simulated annealing and the Steiner problem in graphs, Eng Optim 17 (1991), 91–107.

[3] C.W. Duin, Steiner's problem in graphs, PhD Thesis, University of Amsterdam, 1994.

[4] C.W. Duin and S. Voß, "Steiner tree heuristics—A survey," Operations Research Proc 1993 (DGOR-NSOR), H. Dyckhoff, U. Derigs, M. Salomon, and H.C. Tijms (Editors), Springer, Berlin, 1994, pp. 485–496.

[5] C.W. Duin and S. Voß, Efficient path and vertex exchange in Steiner tree algorithms, Networks 29 (1997), 89–105.

[6] H. Esbensen, Computing near-optimal solutions to the Steiner problem in graphs using a genetic algorithm, Networks 26 (1995), 173–216.

[7] M.R. Garey and D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, SIAM J Appl Math 32 (1977), 826–834.

[8] F.K. Hwang, D.S. Richards, and P. Winter, The Steiner tree problem, Ann Discr Math 53 (1992).

[9] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith, Solving the graphical Steiner tree problem using genetic algorithms, J Oper Res Soc 44 (1993), 397–406.

[10] L. Kou, G. Markowsky, and L. Berman, A fast algorithm for Steiner trees, Acta Inf 15 (1981), 141–145.

[11] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan (Editors), The traveling salesman problem: A guided tour of combinatorial optimization, Wiley, New York, 1986.

[12] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, Inf Process Lett 27 (1988), 125–128.

[13] M. Minoux, Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity, INFOR 28 (1990), 221–233.

[14] J. Pearl, Heuristics, Addison-Wesley, Reading, MA, 1984.

[15] R.C. Prim, Shortest connection networks and some generalizations, Bell Syst Tech J 36 (1957), 1389–1401.

[16] V.J. Rayward-Smith and A. Clare, On finding Steiner vertices, Networks 16 (1986), 283–294.

[17] H. Takahashi and A. Matsuyama, An approximate solution for the Steiner problem in graphs, Math Jpn 24 (1980), 573–577.

[18] P. Winter and J. MacGregor Smith, Path-distance heuristics for the Steiner problem in undirected networks, Algorithmica 7 (1992), 309–327.

[19] A.Z. Zelikovsky, "An 11/6 approximation algorithm for the Steiner problem in graphs," Fourth Symp on Combinatorics, Graphs and Complexity, J. Nesetril and M. Fiedler (Editors), Elsevier, Amsterdam, 1992, pp. 351–354.