Constraint-Based Design of Optimal Transport Elements

Michael Drumheller
The Boeing Company,
Mathematics and Computing Technology,
P.O. Box 3707, M/S 7L-40,
Seattle, WA, 98124
e-mail: michael.drumheller@boeing.com

A large aircraft contains thousands of transport elements, such as tubes, ducts, and wires. Their shape is subject to many constraints, some extrinsic (e.g., obstacle clearance) and others intrinsic (e.g., legal bend angles). A key problem is to design a feasible route that is optimal (e.g., as short as possible). We present an algorithm specialized for metal tubing that allows the user to sketch a route using constraint objects. The user arranges the constraint objects and the system fills in an optimal tube. Trade-offs can be explored rapidly, in terms of quantities of direct engineering interest. This effectively automates a tedious manual design process, saving time and money and producing superior designs. The algorithm has been implemented and tested in a production environment.[1]
[DOI: 10.1115/1.1554698]

1 Introduction

An airliner such as a Boeing 777 has many thousands of *transport elements*, including metal tubes appearing in systems such as hydraulics, pneumatics, fuels, air conditioning, drainage, instrumentation, and fire suppression. (There are many transport elements of other kinds, such as wires, ducts, and hoses, but this paper will focus on metal tubes.)

Transport element design is difficult because, among other reasons, the shape of a transport element is generally subject to multiple conflicting constraints. This is especially true for metal tubes, as we will make clear below. In short, to go from point *A* to point *B*, an arbitrary curve will not suffice. Instead, a valid tube must be a member of a very particular class of curves.

Extrinsic constraints usually arise from the engineering application at hand. The most common types are stay-in or stay-out zones. In contrast, *intrinsic constraints* usually apply to the shape of the transport element itself and arise from manufacturing or assembly considerations. For example, a tube cannot be bent beyond a certain angle without exceeding the mechanical limits of the tube-bending machine, or a hose cannot sustain greater than a certain curvature without kinking. Constraints are not always exact limits, but may have a *preferred* value and a *not-to-exceed* (NTE) value. There is an increasing penalty associated with values passing the preferred value and approaching the NTE value. For example, the preferred minimum bend angle might be 8 degrees, with 4 degrees tolerated, but less than 4 degrees considered infeasible. In addition to constraints, there are multiple *costs* to be minimized, such as weight (length) or flow resistance, and *benefits* (''negative costs'') to be maximized, such as compliance (to ensure ease of installation). The overall goal is to minimize the costs subject to the constraints.

Usually transport elements have low engineering priority. They must adapt continually to the demands of other disciplines. For example, a hydraulics designer might never request that a wing be moved aft, but if the wing is moved aft because of overriding structural or aerodynamic considerations, then hundreds of tubes must adapt accordingly. An automated approach, such as we describe below, could be of great benefit in such re-design scenarios.

1.1 Related Work. Several researchers have addressed tube and pipe routing, e.g., Jain [1], Mitsuta [2], Zhu [3,4], Bohle [5], Satyanarayana [6], and Wangdahl [7]. Automatic routing capabilities also appear in some commercial software products such as [8]. These works generally follow this line: unobstructed space is decomposed into discrete elements; the elements are treated as nodes, and the transitions between them edges, in a graph; an efficient path through the graph is found and the resulting chain of elements is used to derive the transport element shape. The references above emphasize Manhattan-style (orthogonal) routes, which are appropriate for industrial plants or ships, but not for aircraft, where space and weight considerations dictate subtler shapes. Furthermore, these works generally do not account in detail for the strong constraints imposed by manufacturing processes; no one to our knowledge has treated these at the level of detail appropriate for automated tubing design in aircraft. Nevertheless, some aspects of Zhu [3] are very similar to our work (the simplification of background geometry—they use ''virtual sources and sinks''—and abstract shape constraints, e.g., prohibiting the sign of the slope to change on a drainage pipe).

Cagan and Szykman [9] use simulated annealing to synthesize non-orthogonal routes for tubing in industrial plants and mechanical products. Their algorithm captures the inherently discrete nature of bent metal tubes (detailed below), and could in principle deal with intrinsic (manufacturing-based) constraints such as minimum and maximum bend angles, and minimum straight-section lengths. It does not take into account explicitly the circular arcs at bends, which are important to consider in the cramped environs of aircraft. Furthermore, the routes they demonstrate are relatively simple compared to what we address here ([9] demonstrated two or three bends in 3-D, subject only to obstacle constraints, whereas our system is aimed at ''threading'' a tube through a complicated array of constraint objects, not all of which are simple obstacles).

Conru and Cutkosky [10,11] use genetic algorithms to aid in the design of cable harnesses. They not only find efficient paths through free space, but they address the critical problem of refining the paths into physically realizable curves (i.e., satisfying a minimum bend-radius constraint). However, the only kinds of extrinsic constraints they treat are obstacles, and their intrinsic constraints are simpler than required in our setting.

There is a broad literature devoted to superficially related problems such as VLSI layout, water systems, communication networks, etc. Such research tends to focus on 2-D problems, or strictly Manhattan paths. In general, the constraints involved in these areas are not relevant to aircraft tubing.

The field of *mobile robot motion planning* [12], particularly *nonholonomic* motion planning, is directly related to our problem, since a tube is analogous to the path swept out by a spherical mobile robot. A nonholonomic constraint is one that restricts the set of possible motions instantaneously, yet does not reduce the dimensionality of the configuration space. For example, a car is subject to a nonholonomic steering constraint, for it cannot change its orientation without moving forward, but it can *ultimately* assume any position or orientation. Similarly, transport elements cannot be bent arbitrarily sharply, but must be ''steered'' through

Contributed by the Computer Aided Product Development (CAPD) Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received Sept. 2002; Revised Dec. 2002. Associate Editor: K. Lee and N. Patrikalakis

[1]Some aspects are patent-pending in the U.S. (#09/967,784) and other countries.

space. It should be observed that a mobile robot may back up if necessary (e.g., when parallel parking) but tubes enjoy no such freedom. Furthermore, tubes are typically obliged to turn with a *fixed* radius, through a *minimum* arc on each turn, and when a turn is completed the tube must go straight for a *minimum* distance.

Laumond [13] provides an excellent overview of nonholonomic motion planning. The field has deep connections to nonlinear optimal control [14]. Although it focuses on 2-D problems (whereas our problem is 3-D), nevertheless it usefully illuminates the general idea of planning in the face of nonholonomic constraints. The approach introduced in [15] is most similar in spirit to our work. The general program is: (1) find a collision-free path ignoring the nonholonomic constraints; (2) subdivide the path recursively, replacing each section with a valid nonholonomic approximation, until a collision-free path is obtained; (3) refine the resulting path if necessary. (We derived our method independently.) The *elastic bands* method of Quinlan [16] also shares some aspects of our work.

For many readers, the rubric of ''routing'' has a topological flavor and implies discrete decisions, e.g., to pass above not below a certain obstacle, to the left not the right of another, through a particular hole not another, and so on. We call such a set of decisions a *gross route*. We believe ''routing'' is practically a misnomer in our setting, since our method comes into play only after a gross route been determined. Our method *refines* a gross route into a shape that is optimal, manufacturable, and pleasing to a human designer.

The chief differences between our method and previous work may be summarized as follows: our method works in 3-D; we introduce special *constraint objects* that are adapted to the complex environment of aircraft; we treat tubing geometry in great detail (taking rounded corners into account explicitly); and, we provide a means of automatically exploring trade-offs between various designs.

1.2 Current CAD Approaches. In the early days of aircraft design, tubes were designed by hand-bending model tubes against a wooden mockup. Nowadays tubes are usually designed electronically with a computer-aided design (CAD) system. Here we describe how this is currently done. We introduce some important background and terminology first.

Geometry of Bent Metal Tubes. Geometrically, a transport element is a 3-D curve, called a *centerline*, along which is swept a cross-sectional shape (usually a circle). Metal-tube centerlines belong to a peculiar class of curves—alternating straight sections and circular arcs—due to their manufacturing process: tubes are formed from straight pieces of stock by introducing a series of circular bends using a special tube-bending machine. A detailed description of the process may be found in [17] (see also [18,19]). Essentially it consists of ''shooting'' out a length of straight stock, bending it around a circular die, and rotating the stock around its longitudinal axis. This process may be repeated several times.

The straight sections of a tube are subject to a minimum-length (MINLEN) constraint because the tube-bending machine can grip the tube only at straight sections. The bends are subject to a minimum-angle (MINANG) constraint, since small bends are difficult to produce (the tube may spring back elastically). Bends are also subject to a maximum-angle (MAXANG) constraint, chiefly because large bends are likely to cause the tube to collide with the tube-bending machine itself. (Placing an upper limit on the bend angle is a mere heuristic; it cannot generally prevent such collisions.) Usually all bends in a tube have the same radius r, since varying the radius from bend to bend requires a die change, which is costly. The object of design is a *tube run*, which consists of several individual tubes connected in a series. The tube run is usually conceived and designed as a whole, and later divided into sections for assembly. In practice r is relatively free to change from section to section. (In this paper we assume it is constant.)

Referring to Fig. 1, the ends of a tube are called the *A-end* and the *B-end*. The direction *A*-to-*B* is called *downstream* and *B*-to-*A* *upstream*. The (imaginary) intersections $\mathbf{N}_i \in R^3$ of adjacent straight sections are called *nodes*. The arcs with radius r terminate at *tangents* $\mathbf{T}_i^A$ and $\mathbf{T}_i^B$. The length of the straight section at A is called the *standoff* s_A, and s_B at B. The length l_i of all other straight sections is bounded below by $l_{\min}$. Bend angles α_i obey $\alpha_{\min} \leqslant \alpha_i \leqslant \alpha_{\max}$. H_j is a *hardpoint* (e.g., a clamp). The straight section through H_j has orientation limits and must extend upstream by at least O_j^A and downstream by O_j^B (to provide clearance for installation or sliding). $O_j^A + O_j^B$ may be less than $l_{\min}$. The term *route* denotes an ordered set of nodes $(\mathbf{N}_1, \ldots, \mathbf{N}_n)$. A *centerline* is obtained from a route by interpolating straight sections and arcs. A swept disk of diameter D defines the final tube shape. Typically $l_{\min} \approx r \approx 3D$, $\alpha_{\min} \approx 8°$, and $\alpha_{\max} \approx 120°$.

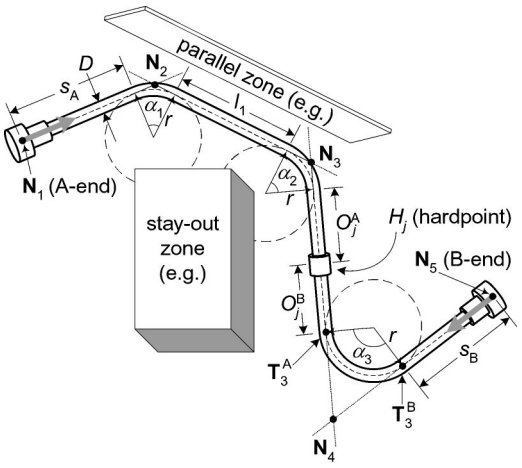

Fig. 1 Tubing geometry

Metal tubes are essentially rigid, so their shape must be specified exactly. It is not sufficient, for example, to simply cut a rough length of material and rely on a technician to adjust the shape on the factory floor. (In some cases, a small degree of *preloading*, or elastic deformation, may be introduced during assembly, as when following the gentle curve of a wing surface. Preloading also stems from unavoidable variability in the fabrication process, which Wei [17] treats in detail. We do not treat these issues in this paper.)

Node-Based Design; the Linkage Viewpoint. The nodes and the bend radius constitute a very compact shape representation. It is so convenient that it is often used as a basis for design, which can be problematic. For example, in Fig. 2 a minimum standoff (MINOFF, typically $\approx 3D$) of s_A is needed at A. This is achieved by placing $\mathbf{N}_2$ sufficiently far from $\mathbf{N}_1$. However, if the next downstream straight section is moved, as when passing from (1) to (2) in Fig. 2 with $\mathbf{N}_2$ fixed, then MINOFF may be violated. This illustrates how it is not generally possible to simply design a tube in terms of nodes, and then blindly interpolate the arcs and straight sections.

Current CAD systems provide several aids for designing tubes, but they do not escape essentially node-based design. CATIA [20], for example, provides a ''compass'' (a movable local coordinate

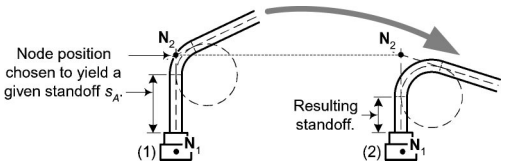

Fig. 2 Nodes make poor control points

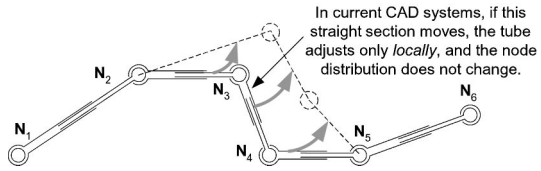

Fig. 3 Linkage viewpoint

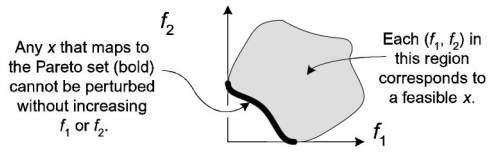

Fig. 5 The Pareto set

system) that can be positioned relative to the background geometry. The user draws a tube one section at a time, using the compass to guide each section. The system automatically checks the manufacturing constraints *after* a new section is added. (Such local, *post hoc* correctness checking tends to prevail in CAD systems, whereas our method represents a global, correct-by-construction approach.) The result is an inherently manual, incremental, and node-based process.

Some other forms of constraint management are available, e.g., the user may specify that part of a tube should maintain a given separation from a background element (e.g., [21]). However, only relatively simple constraints, involving a small number of nodes or straight sections, may be applied. Systems that work this way (notably [18]) essentially view a tube as a linkage made up of ball- and telescope-joints, as in Fig. 3, but one that responds only locally to user input or applied constraints. As we will show below, our method adjusts the entire linkage simultaneously.

Node Distribution. Current CAD systems regard the *node distribution* (the number of joints in the ''linkage,'' and their ordering relative to external constraints such as obstacles or clamps) as a fixed entity that the user must design explicitly. This viewpoint can be problematic. For example, in Fig. 4(*A*), the bulkheads are parallel, and the tube is straight. In (B) the top bulkhead has been rotated very slightly, and the tube consequently has a very small bend, but the bend violates the MINANG constraint. The three-bend configuration in (C) is needed to accommodate the seemingly trivial change in the background geometry, since only by introducing ''extra'' bends can the MINANG constraint be satisfied (assuming no elastic deformation). Evidently requiring the user to design a fixed linkage beforehand can preclude automatic redesign when the background changes. What is needed is a system that can automatically re-design the linkage.

1.3 Trade-Offs. Even with the CAD aids described above, the tubing design process still relies heavily on trial and error. The engineer designs the tube by manually manipulating the nodes. If the constraints seem to preclude a satisfactory design, then he relaxes one or more constraints and starts over. The choice of which constraints to relax, and in what order, and by what amount, is typically based on intuition or experience.

The problem is that constraints tend to interact in inconvenient ways. Setting one at its preferred value tends to drive the others toward their not-to-exceed (NTE) values. As with a radio button, pushing one down forces the others up, with no available setting in between. An in-between configuration may be what the designer wants, but obtaining it by manipulating the geometry explicitly can require excessive trial-and-error work. Our method, in contrast, exposes in-between configurations directly by introducing ''soft constraints.'' A soft constraint assigns a cost to values between the preferred and NTE values. The overall objective includes these *violation costs*. The user can explore trade-offs by adjusting the weights, not the geometry. The geometry (potentially even the node distribution) adjusts itself automatically.

One may ask, ''What are the 'correct' weights for balancing, say, MINANG violations against flow resistance?'' Such a question is not well posed. The best that can be done is to expose as much of the *Pareto set* as possible (see Fig. 5). This set contains the *efficient* designs (those that cannot be perturbed without increasing some costs) [22]. The user must choose a *particular* efficient design, but with our method he can at least restrict his attention to ''efficient design space.'' In our opinion this is an important advance over current CAD capabilities.

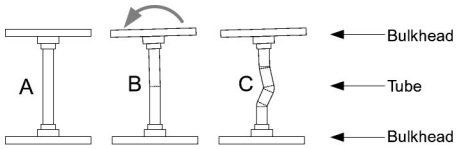

Fig. 4 A *single* node distribution may not be able to accommodate all changes in the background

2 Method

2.1 Overview. Our method provides *constraint objects* in terms of which a designer can ''sketch'' a gross route. Constraint objects might be local in nature, such as stay-in or stay-out zones, or zones in which a given orientation is required. They may also be global, enforcing, e.g., planarity, slope restrictions, and so on. The user may also specify relationships between disparate parts of the tube, so that, e.g., the orientation at a certain point matches that at another point, without specifying the orientation *per se*. The user chooses parameterized constraint objects from a ''palette'' and places them in the environment, producing a sketch of the desired route. The system then fills in an optimal, manufacturable tube.

Constraint objects act as *surrogates* for real background geometry elements. This ''surrogate background approach'' allows (and requires) the user to ''mark up'' the background with constraint objects that stand in for real geometry. There are two reasons for this: First, background models generally have more detail (e.g., rivets, fillets, etc.) than is needed to drive the design of transport elements and it is advantageous to suppress such detail for the sake of computational efficiency. Second, the background geometry may not explicitly represent the constraints that the designer knows are important. For example, the designer may know that when a transport element runs near a certain fixture, it should do so with a particular orientation and offset, but this knowledge is simply not represented in the background model.

2.2 Constraint-Object Types. It is convenient to define three types of constraint objects: *targets, section constraints,* and *constraint relationships*.

Before explaining all these types, it will be helpful to explain *hardpoints* first. A hardpoint defines the position and orientation of a straight section of a tube, and typically represents the location of a clamp. Hardpoints are convenient design primitives: the user can specify an *A*-end, a *B*-end, and a sequence of intermediate hardpoints, and a CAD system can fill in the straight sections and arcs (if possible).[2] (This is nearly equivalent to node-based design, since the intersections of hardpoint-lines form nodes.) Even when using only hardpoints, filling in the straight sections and arcs *optimally* (e.g., with the shortest path) for an entire tube in the face of MINANG, MAXANG, and MINLEN constraints is generally non-

[2]A system at Boeing called KIRTS is capable of sophisticated hardpoint-based design; see *Acknowledgments*.

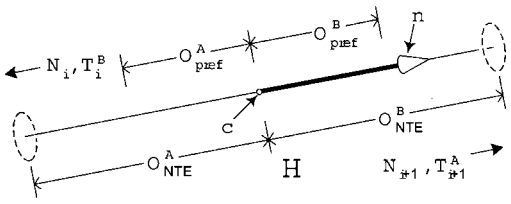

Fig. 6 Model of a hardpoint

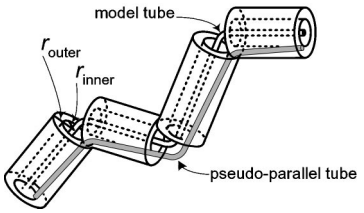

Fig. 8 Sleeves for pseudo-parallel routing

trivial. Subsections of a tube can interact, albeit weakly, across hardpoints via the MINLEN constraint, so the entire tube run must be considered as a whole.

A single hardpoint implies several ''primitive'' constraints and costs. Figure 6 depicts a model of a hardpoint H that clamps a straight section $\overline{\mathbf{N}_i\mathbf{N}}_{i+1}$. H has a center $\mathbf{c}$, orientation $\mathbf{n}$, preferred upstream (downstream) offset $O^A_{\text{pref}}(O^B_{\text{pref}})$, and not-to-exceed upstream (downstream) offset. O^A_{NTE} (O^B_{NTE}). H gives rise to several constraints: $\overline{\mathbf{N}_i\mathbf{N}}_{i+1}$ must intersect $\mathbf{c}$; the angle between $\overline{\mathbf{N}_i\mathbf{N}}_{i+1}$ and $\mathbf{n}$ must be zero; and, the distance from $\mathbf{c}$ to $\mathbf{T}^B_i$ $(\mathbf{T}^A_i)$ must not fall below $O^A_{\text{NTE}}(O^B_{\text{NTE}})$. (These are ''primitive'' in that they are conceived in terms of the simplest available geometric entities—points, lines, etc.). These can carry soft-constraint costs as well. Thus, even a simple hardpoint is a fairly complicated ''bundle'' or ''macro'' of constraints and cost functions.

Targets. *Targets* may be viewed as ''generalized hardpoints.'' They serve as waypoints in a gross route. Figure 7 shows four commonly used targets. (A) is an *A*-end or *B*-end with a finite angular tolerance. (B) is a rectangular region inside of which the tube must lie parallel to the arrow. (*H* indicates the *eventual* location of a hardpoint.) (C) is a rectangular stay-in zone through which the tube must pass with angular limits indicated by the cone shape. It is different from (B) because the range of motion of the hardpoint is substantially perpendicular, not parallel, to the tube direction. (D) is a 3-D stay-in zone (of which (B) and (C) may be viewed as special cases). Targets such as (B), (C), and (D), in which a hardpoint is allowed to ''slide'' within an admissible zone, are called *sliding hardpoints*. Such targets effectively let the user say ''Place a hardpoint *somewhere* in this region, but defer finding its *exact, optimal* location until later.'' This reflects a general theme in our method, namely to reduce the user's workload by demanding the least specific input possible. Notice that (B) and (D) require an extra parameter, besides the relevant node coordinates, to locate *H*.

If hardpoints were the only targets allowed, then a good node distribution would follow trivially from only local considerations: the number of nodes between adjacent hardpoints is predetermined by their relative configuration (in reasonable designs the number must be 0, 1, or 2). In contrast, with the ''loose'' targets of Fig. 7, the number of nodes between targets may vary according to the *overall tube shape* (see Fig. 14 for an example).

Figure 8 shows a more unusual target type. Each ''sleeve'' surrounds a portion of an existing transport element. The new transport element is constrained to not approach the existing transport element closer than r_{inner}, nor to be farther away than r_{outer}. This allows the user to route a new tube ''pseudo-parallel'' to an existing one.

Other targets are more abstract, e.g., an infinite ''slab'' of space in which the transport element must run parallel to a certain plane, but with no other restrictions. Not all targets need to have a hardpoint associated with them. Each target explicitly influences a fixed number of nodes.

To gain an idea of the complexity of the primitive constraints that a target (and constraint objects in general) can give rise to, consider Fig. 9. It illustrates how a single target (in this case an irregular hoop-shaped stay-in zone) gives rise to considerably more primitive constraints and costs than a simple hardpoint. These primitive constraints involve complicated expressions: the distance O^A_j between the hoop plane (defined by $\mathbf{C}_j$ and normal vector $\mathbf{m}_j$) and the tangent $\mathbf{T}^B_i$ is constrained above a not-to-exceed value, and carries a decreasing cost as it approaches a preferred value. Given a bend radius of r, O^A_j has the form

$$O^A_j = -\mathbf{m}_j \cdot \left(\mathbf{N}_i + r\mathbf{n}_i \sqrt{\frac{1-\mathbf{n}_i \cdot \mathbf{n}_{i-1}}{1+\mathbf{n}_i \cdot \mathbf{n}_{i-1}}} - \mathbf{C}_j \right)$$

where $\mathbf{n}_i = (\mathbf{N}_{i+1} - \mathbf{N}_i)/|\mathbf{N}_{i+1} - \mathbf{N}_i|$. The soft constraint cost is $(O^A_j)^2$; similarly for $(O^B_j)^2$. The expression for the distance d_i between $\mathbf{edge}_i$ and $\mathbf{C}_j$ involves a term accounting for the effective elongation of the tube diameter due to its generally oblique relation to the plane. The d_i are constrained to be non-negative, keeping the tube inside the hoop; the d_i^2 form soft-constraint costs. The direction of penetration, characterized by γ_j and δ_j, is also constrained, and also contributes soft-constraint costs. Notice that *inequality* constraints play a strong role, in contrast to typical parametric CAD systems, which expose only equality constraints.

Section Constraints. These are so named because they apply to the section of a tube between two targets. A section constraint may have an obvious geometrical manifestation (e.g. a stay-out zone; see Fig. 15–18), or it may be more abstract, imposing con-

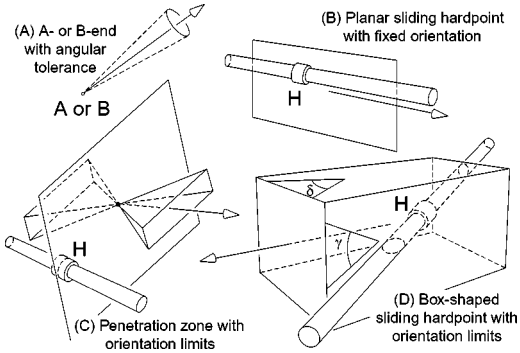

Fig. 7 Some available targets

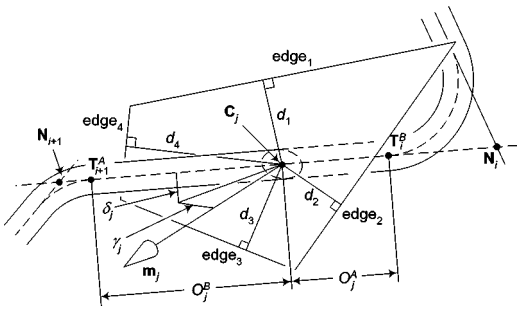

Fig. 9 A single high-level constraint implies many "primitive" constraints

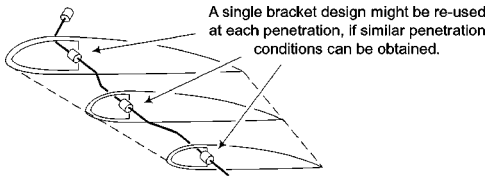

Fig. 10 Constraint relationships may enforce similar conditions at disparate locations

straints on, or associating costs to, factors such as slope, overall orientation, or planarity. (Slope constraints may be needed to avoid trapping moisture; planarity may be desirable because planar designs can be used on either side of an airplane.) Since a section constraint applies to everything between two targets, and since it is not known *a priori* how many nodes there will be between any two targets, it is therefore not known *a priori* how many nodes a given section constraint will affect. For example, a stay-out zone might lie between an *A*-end and a *B*-end, and it may be possible for the transport element to circumvent the stay-out zone with only two bends, in which case the separation constraints that cause the desired stay-out behavior apply to three straight sections and two arcs (i.e., to two nodes). However, if more bends are required, then the stay-out zone might affect more nodes.

Constraint Relationships. These are ''meta-constraints'' that establish relationships between, or introduce new constraint- or cost-functions of, two or more targets or section constraints (usually targets). For example, it may be advantageous for a transport element to penetrate a sequence of stay-in zones at the *same* angle, or with the *same* offset from the edges of the zones—without specifying what that angle or offset should be. This idea is illustrated in Fig. 10. Another use for constraint relationships is to limit or regulate the distance between sliding hardpoints. The user can place a sequence of sliding hardpoints (recall Fig. 7(*D*)) and let the algorithm produce the best inter-hardpoint spacing. Constraint relationships can introduce an arbitrary degree of nonlocality into the design problem.

2.3 Optimization Approach. A key aspect of our method is automatically finding a suitable *node distribution* (see §1.2). We take a heuristic approach to this problem, consisting of first finding a ''reference route'' that likely resembles, in a rough sense at least, an optimal route, and then using this reference route as a guide for estimating the final node distribution.

General Pattern; Overall Cost. The general pattern of our algorithm is to formulate a series of continuous sub-problems, each according to a particular trial node distribution. The independent variables in each sub-problem are the node coordinates, plus extra parameters determining the exact hardpoint locations (in those targets for which the hardpoint location is adjustable). Corresponding to each sub-problem, there is a continuous cost f that depends only on the continuous node positions, and an overall cost Φ, which comes from additively augmenting f with some function g of the discrete node distribution. For example, g might be the number of nodes (reflecting a desire to minimize the number of bends explicitly). Generally $\Phi=\beta_1 f+\beta_2 g$, where β_1, $\beta_2\geqslant 0$ are user-adjustable weights.

Other Viewpoints: Mixed-Variable Optimization. The problem may be viewed as an instance of *mixed-variable optimization*, since some independent variables are discrete and others continuous (see [23] for other examples). Our goal is to simultaneously determine a node distribution (the *discrete* number of nodes in each inter-target gap) *and* the exact node positions (continuous quantities). Mixed-variable problems can sometimes be approached using *mixed-integer programming*, in which the discrete variables are relaxed to continuous variables that must take on integer values only at a final solution. A mixed-integer formulation could run along the following lines: Corresponding to each node $\mathbf{N}_i$ there would be a vector $\mu_i(x)$ of constraints describing the MINANG/MAXANG conditions, and another vector $\sigma_i(x)$ describing ''no-bend conditions'' (collinearity). A standard device for ''activating'' or ''deactivating'' $\mu_i(x)\leqslant 0$ or $\sigma_i(x)\leqslant 0$ is to introduce variables y_i, z_i and the modified constraint equations

$$\mu_i(x)-My_i\leqslant 0$$

$$\sigma_i(x)-Mz_i\leqslant 0$$

with y_i, $z_i\in\{0,1\}$, $y_i\neq z_i$, and $M\gg 1$. Then one would apply, for example, a branch-and-bound algorithm, as in [24]. We would anticipate several problems: First, at each deactivated bend all constraint and objective functions are invariant with respect to translation of a node along the centerline, so the problem would be ill-conditioned in the continuous subspace. Second, the number of continuous variables would be constant and large. (In our formulation it varies as the solution progresses, and is kept as small as possible.) Third, the MINLEN constraint would be extraordinarily cumbersome to implement, since it must take into account contiguous occurrences of the no-bend conditions.

Why Not a Straightforward Continuous Optimization? The following purely continuous program might seem reasonable: (1) introduce an ''overpopulation'' of nodes; (2) optimize their locations continuously; (3) allow bends or straight sections to vanish (or coalesce) as necessary. However, the MINANG and MINLEN constraints, which are *essential* to the problem, foil such an approach, since they *explicitly forbid any bends or straight sections from vanishing continuously*. Thus no purely continuous optimization method can solve the problem by itself. Discreteness, which is generally known to make optimization difficult, is an unavoidable aspect of our problem.

Graph Search. It would be possible to decompose space discretely and apply graph search procedures such as Dijkstra's algorithm or A* search (following the lines of the more conventional path planning literature; see §1.1). Such a *purely discrete* approach cannot provide exactness, which we consider an important goal for very detailed tubing design. It can only provide an approximation whose precision accords with the fineness of the discretization.

2.4 Initial Guess, Easy Pass. With the above viewpoints as background, we proceed to describe our optimization approach in detail.

Initial Guess. The first step is to find a ''reference'' route that likely captures the main features of the final route, i.e., it ''hits'' all the targets and obeys the extrinsic constraints, and thus tends to indicate the rough locations of large bends, small bends, long or short straight sections, etc. The order of the targets is determined automatically, but the user can override this ordering if desired. As suggested in the ''straightforward continuous'' approach discussed above, we introduce a gross overpopulation of nodes through an *initial guess* procedure: each pair of adjacent targets generates a guess of a small number of nodes between them (typically two, but special configurations may give rise to one or zero). Section constraints may ''sprinkle'' additional nodes into their associated inter-target gaps. For example, a stay-out zone between two hardpoints typically adds three extra nodes between those already introduced by the targets, to ensure enough degrees of freedom to circumvent the stay-out zone.

Figure 11 illustrates the initial-guess procedure. The nodes in black are guessed by nearby targets. Each black node occupies a predetermined position relative to the two nearest targets, regardless of other constraint objects. The white nodes arise from the stay-out zone. The initial guess is not necessarily feasible: notice, for example, the stay-out violation in Fig. 11. Therefore it does not necessarily provide a good indication of the eventual overall

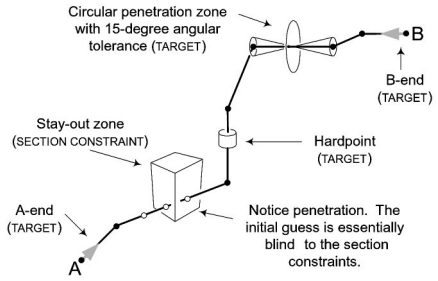

Fig. 11 Initial guess

shape. Typically, the initial guess includes many more nodes than are eventually needed in an optimal route; hence the term ''gross overpopulation.''

Easy Pass. In this phase we optimize the continuous node locations of the initial guess, relaxing the intrinsic constraints MINANG, MAXANG, and MINLEN, and using a greatly reduced bend radius. The goal is simply to run a ''thread'' through all the targets that obeys all the section constraints and constraint relationships, and thereby provides a better indication of the eventual overall shape. The resulting centerline is called the *easy pass*. It is critical to observe that if we attempted to thread an overpopulation of nodes enforcing the MINANG, MAXANG, and MINLEN constraints then very likely either (a) no feasible configuration would exist, or (b) the resulting route would be wildly different (i.e., containing loops, etc.) from that obtained with a good node distribution. Therefore, a key point is to relax the intrinsic constraints during the easy pass optimization. Figure 12 illustrates the easy pass. (The dotted lines and other annotations in Fig. 12 are explained below.) Notice that even the easy pass is not *guaranteed* to succeed, as it is always possible to arrange the constraint objects in a pathologically infeasible way regardless of the particular constraint limits.

Salience Ranking. The nodes in the easy pass are ranked by *salience*, which is defined as the deviation of a node at the moment it appears as a breakpoint in a classical iterative endpoint-fitting algorithm (see [25]). This generates a sequence of straight-line approximations to the easy-pass route $(\mathbf{N}_1, \ldots, \mathbf{N}_n)$ as follows: We find the node $\mathbf{N}_i$ whose perpendicular deviation $d_{\mathbf{N}_i}$ from the line L_{1n} through $\mathbf{N}_1$ and $\mathbf{N}_n$ is greatest. The route is broken at $\mathbf{N}_i$ to form two sub-routes $(\mathbf{N}_1, \ldots, \mathbf{N}_i)$ and $(\mathbf{N}_i, \ldots, \mathbf{N}_n)$ with corresponding lines L_{1i} and L_{in}. This is done recursively, halting when the sub-routes contain only two nodes (see **U**, **V**, and **W** in Fig. 12). By this means each node is eventually assigned a salience $d_{\mathbf{N}_i}$.

2.5 Choosing the Best Nodes from the Easy Pass. The next two phases heuristically ''filter'' the easy pass nodes, based

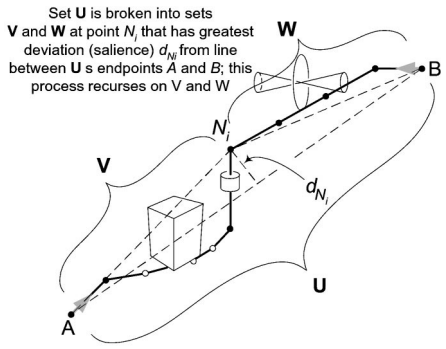

Fig. 12 Salience ranking and easy pass

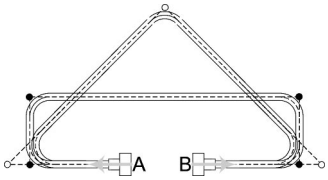

Fig. 13 Potential benefit of look-ahead

on their saliences, in order to determine which ones form a good node distribution.

Terminology: Given a route $R=(\mathbf{N}_1, \ldots, \mathbf{N}_n)$ with saliences $d_{\mathbf{N}_2}, \ldots, d_{\mathbf{N}_{n-1}}$ (terminal nodes are never assigned a salience), consider a route S obtained by deleting some or all of the internal nodes $\mathbf{N}_2, \ldots, \mathbf{N}_{n-1}$. A deleted node is said to be *re-admitted* to S if it is inserted into S in the same relative position that it had in R. A node is re-admitted only if it has the greatest salience of all nodes that have not yet been re-admitted. Therefore, given S created from R, a *re-admission* operation on S unambiguously adds a predetermined node. Given a route R, a node is *suppressed* by deleting it. Some nodes in R may be declared *critical* (see below) and are never suppressed. A node is generally not suppressed unless it has the least salience of all nodes not yet suppressed. Therefore, given a route R, a *suppression* operation on R unambiguously deletes a predetermined node.

Re-admission on a route S is defined only in the context of some original route R from which S was obtained by deleting nodes. Suppression, in contrast, on any route is defined in terms of just the route itself.

An *optimized re-admission* operation is a re-admission followed by an attempt to optimize (continuously, using conventional optimization techniques) the node locations and the parameters that determine the sliding hardpoints' exact locations. (The continuous optimization process is described below.) An *optimized suppression* operation is a suppression followed by a similar optimization attempt. An optimized re-admission or optimized suppression is said to fail if no feasible point can be found, i.e., if the node coordinates and other parameters cannot be adjusted in a continuous sense to achieve feasibility. From a performance point of view, the algorithm exploits the fact that conventional continuous optimizers typically determine non-feasibility quickly.

Forward Pass with Look-Ahead. A *forward pass* is executed as follows: From the easy-pass route $E=(\mathbf{N}_1, \ldots, \mathbf{N}_n)$, nodes $\mathbf{N}_2, \ldots, \mathbf{N}_{n-1}$ are deleted to obtain the trivial trial route $(\mathbf{N}_1, \mathbf{N}_n)$. An attempt is made to optimize the locations of $\mathbf{N}_1$ and $\mathbf{N}_n$. With high probability this optimization will fail, since the only routes that contain two nodes are those connecting two collinear end-fittings, which is a rare scenario. If it fails, optimized re-admissions are applied, in order of decreasing $d_{\mathbf{N}_i}$, until either (a) a re-admission is successful, producing a route F, or (b) there are no more nodes to re-admit. In case (b) complete failure is declared. In case (a), the overall cost is recorded as C_F. It is advantageous to ''look ahead'' to see whether adding nodes can produce a significantly better route. Figure 13, for example, illustrates how a four-bend (six-node) route may be superior (in terms of, say, length) to a three-bend (five-node) route. Look-ahead consists of further optimized re-admissions until one is successful or no nodes remain. The cost of the first feasible look-ahead route F' (if one exists) is recorded as C'_F. If $C'_F \leqslant (1-\theta)C_F$, $0 \leqslant \theta \leqslant 1$ (solutions are improving ''quickly'') then the result G of the forward pass is taken to be F', with cost $C_G = C'_F$. Otherwise $G=F$ and $C_G = C_F$. The parameter θ (typically 0.1) is essentially a threshold on the rate of change of the overall cost with the number of nodes (it effectively controls ''coarseness'': a final route will have fewer nodes according as θ is large). Look-ahead could be extended beyond adding just a single node, but this has not proved necessary. Notice that on the forward pass, the saliences $d_{\mathbf{N}_i}$ are always taken from the easy pass.

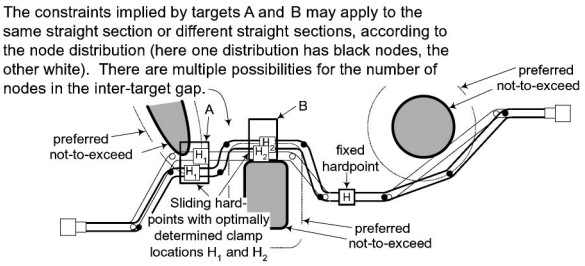

Fig. 14 Alternative node distributions

Backward Pass. Frequently some nodes may be re-admitted in the forward pass that are not strongly needed. This is related to a phenomenon observed in certain regression schemes [26,27], where it is found effective to progressively add complexity to a model until a good fit is obtained, and then progressively reduce the complexity until the fit worsens significantly. Analogously, it is often possible to delete a few nodes from the result of the forward pass without degrading the route significantly. This reflects the fact that (a) salience is merely an ordering heuristic, not a perfect indicator of re-admission priority, and (b) after a node is re-admitted, all nodes are adjusted by a continuous optimization, so some nodes generally end up in different locations from those they held in the easy-pass route *E*.

A *backward pass* is executed as follows: First, the saliences $d_{\mathbf{N}_i}$ are re-calculated on *G*. Optimized suppressions are applied to *G*, in order of increasing salience, until one is successful or there are no more nodes to suppress. Each time an optimized suppression fails, the node that was to be suppressed is declared *critical*. If an optimized suppression succeeds, producing a route *B* with cost C_B, and if $C_B \leqslant C_G(1+\sigma)$, $\sigma \geqslant 0$ (the solution does not degrade "too quickly"), then *G* is set to *B*, saliences $d_{\mathbf{N}_i}$ are re-calculated, and the process recurses with the new *G*. Notice that in the backward pass the saliences are continually re-calculated, whereas in the forward pass the saliences from the easy pass are reused. The following refinement is effective: Upon failure of an optimized suppression, the node $\mathbf{N}_i$ is not immediately declared critical. Instead, another form of look-ahead is applied, to address what we call "pairwise criticality": a node might not be critical if the next-least salient node $\mathbf{N}_j$ is *also* suppressed. More precisely, a route G' is formed, equal to *G* with $\mathbf{N}_i$ deleted. An optimized suppression is applied (on the next-most-salient node $\mathbf{N}_j$) to G', and if it succeeds then *both* $\mathbf{N}_i$ and $\mathbf{N}_j$ are deleted. This idea could be extended beyond two nodes, but this has not proved necessary.

2.6 The Continuous Sub-problems. The forward-pass/backward-pass procedure yields the final node distribution. During those passes many different node distributions must be optimized in a continuous sense. Figure 14 illustrates how different node distributions imply fundamentally different continuous sub-problems. For these we use NPSOL [28], a well-known augmented-Lagrangian-based algorithm (Gill [29]). NPSOL exposes the following interface directly to the programmer:

$$\begin{aligned} &\min \Phi(x) \quad \text{subject to} \\ &\alpha_l \leqslant C(x) \leqslant a_u \\ &\beta_l \leqslant Ax \leqslant \beta_u \\ &\gamma_l \leqslant x \leqslant \gamma_u \end{aligned} \tag{1}$$

where α_l, α_u are lower and upper bound vectors on the general nonlinear constraints *C*; β_l, β_u are bounds for the linear constraints represented by the matrix *A*; and γ_l, γ_u are simple bounds on the independent variables *x*. In the continuous sub-problems *x* consists of the node coordinates plus parameters that characterize non-node features (sliding-hardpoint locations) that are optimized concomitantly, i.e.,

$$x = (x_1, y_1, z_1, \ldots, x_n, y_n, z_n, \pi_1, \ldots, \pi_m)$$

Here x_i, y_i, and z_i correspond to node $\mathbf{N}_i$. The π_i locate sliding hardpoints within their admissible zones (recall Fig. 7(*D*)). Space limitations prohibit writing out the full objective function explicitly, even for the simplest case, but Fig. 9 and its associated text convey how complex it can be for even a single target. Many primitive quantities (analogous to those of Fig. 9 but adapted of course to the particular constraint objects in question) contribute entries to the $C(x)$ vector, rows to the *A* matrix, or members to the simple-bounds list, according to their mathematical form. Each quantity may also contribute a cost component to the continuous objective function. Thus the continuous objective function is a weighted sum of "primary" costs such as length, pressure drop, stiffness, etc., and "secondary" costs due to soft-constraint violations. The weights are user-adjustable (recall §1.3)). Any subset of costs can be combined so as to minimize the maximum of the elements. Such a subset is called a "minimax group," and is enabled by the AUTOOPT software described below. For example, all minimum bend angle costs can be combined as a minimax group, which is then included as an additive cost component. This can have a startling effect, since all members of such a group will tend to seek the same maximum value. So far this capability has not been used enough to judge what situations it might be preferred in.

Many of the cost functions involved do not satisfy certain common requirements of optimizers (e.g., C^2 continuity). Stay-out zones in particular involve **dist**(A,B) (the minimum distance between two point sets), which is usually only singly differentiable. Such inadequacies prove inconsequential in practice.

Utilizing a New Optimization Software Architecture. Each time a new node distribution is considered, the form of the corresponding continuous sub-problem generally changes drastically. This was illustrated in Fig. 14. Here we discuss the software implications of this complication. In short, a large part of our code is devoted to setting up, at run-time, the complicated continuous sub-problems for the trial node distributions. The raw interface of Eq. (1) is extremely hard to use for this directly. The essential difficulty is that variables are named only by their indices in arrays. This creates difficult bookkeeping problems when formulating new problems on the fly. To solve this problem we developed a new object-oriented software package, called AUTOOPT, which serves as a "wrapper" for any NPSOL-like optimizer. AUTOOPT allows the programmer to create individual constraints and objectives in a dynamic, run-time manner, using an abstract namespace. The user can program his formulas using arbitrary variable names. Formulas are assigned to objects, and can be mixed and matched at will; AUTOOPT arranges and accesses them in the form and order expected by the underlying optimizer. The key feature of AUTOOPT is an abstract namespace that can be programmatically manipulated at run-time. This provides many of the advantages of mathematical programming languages such as AMPL or GAMS, but rather than emphasizing human readability, AUTOOPT emphasizes machine manipulability. In this sense AUTOOPT is similar to the commercial products [30,31] but is, in our opinion, more general than either.

3 Results

Figure 15 shows two tubes in an airliner wheel well. Tube **A** circumvents various stay-out zones around structural elements. It also passes through two sliding hardpoints; the final hardpoint positions were automatically chosen within the white-outlined rectangular regions. The inter-hardpoint spacing can in general be regulated (limited or equalized) by appropriate constraint relationships. Tube **B** obeys most of the same section constraints as tube **A**, but is guided in a "pseudo-parallel" route by the sleeve objects surrounding tube **A**.

Figure 16 illustrates how constraint objects, which are themselves "macros" of primitive constraints, can in turn be combined

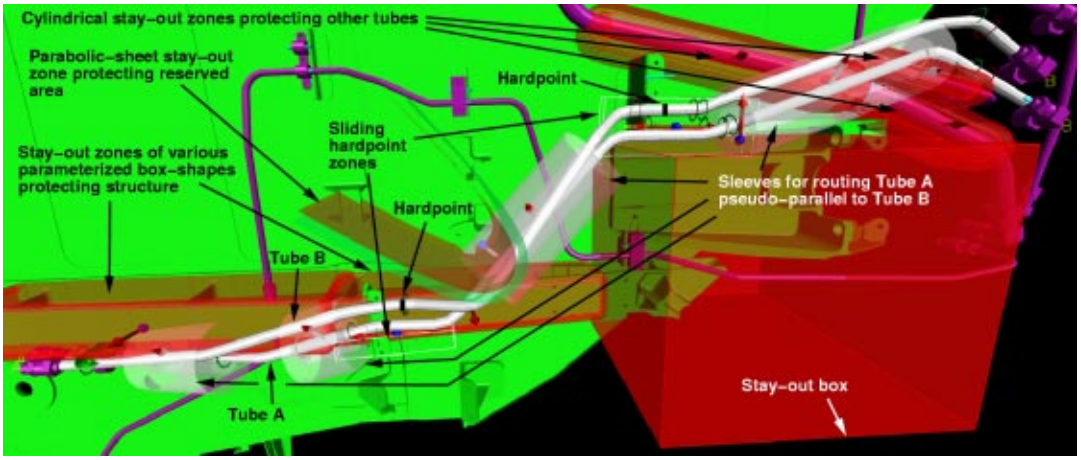

Fig. 15 Pseudo-parallel tubes in a wheel well

into larger macros. Three sliding hardpoints are arranged in a **U** shape to produce an *expansion loop* (such loops are used to give a tube sufficient compliance to withstand installation or thermal stresses, vibration, etc.). Figure 16 demonstrates how a tube can ''re-design itself'' in response to changes in the background geometry (it shows the loop before and after the introduction of two stay-out zones). Notice that the objective function could incorporate an explicit probability of rejection due to insufficient compliance, as in Wei [17].

Figure 17 shows a route through two rectangular hoops, each of which has a large range of admissible angles (recall Fig. 7(*C*)), with two stay-out zones in between. Hoops are typically superimposed, as in this example, on stiffening webs or bulkheads to indicate regions of admissible penetration. Figure 17 also includes some stay-out zones protecting some of the background structure. Tube **A** penetrates stiffeners 1 and 2 at significantly different angles, with respect to their local coordinate systems (**x1**, **y1** and **x2**, **y2**). The angles are indicated by the dotted lines **A1** and **A2**. Tube **B**, in contrast, is subject to all the same constraints as tube **A**, with an additional *constraint relationship* (§2.2) imposed, namely that the incidence angles should be the *same* at both penetrations (without specifying the angle *per se*), thus facilitating the re-use of a clamp or bracket design. Observe that the desired behavior (see **B1** and **B2**) is obtained not merely by a continuous adjustment, but by changing the node distribution (the ''linkage design,'' §1.2) and this is done automatically.

Performance. Table 1 shows typical timings for the various algorithmic phases (§2) for a tube passing through a sliding hardpoint and a hoop, and avoiding a stay-out zone (see Fig. 18). (All results are with an SGI workstation.) Notice that infeasibility is typically determined very quickly. The typical time to evaluate the objective and all the constraints once is about 0.4 milliseconds (This dominates the continuous optimization time.) Longer tubes currently take much more time; e.g., in Fig. 15 tubes **A** and **B** had 14 and 19 initial nodes, 9 and 14 final nodes, and took 71 and 48 seconds, respectively. (The typical time to evaluate the objective and constraints was about 2.5 milliseconds.) The increase in time stems not only from (a) the greater number of independent variables and extrinsic constraints, but (b) some of the continuous sub-problems tend to be ill-conditioned, i.e., the continuous optimizer is forced to search in a ''valley'' (a local symmetry of the objective with respect to some perturbations of the independent variables). Such regions tend to defeat many stopping criteria, causing continuous optimizers to get ''bogged down'' in a realm of diminishing returns.

There are many performance optimizations that we are planning to implement but have not tried yet. We give a brief discussion of some of them here: (1) Regarding problem size, there are many opportunities for eliminating variables (e.g., replacing equality

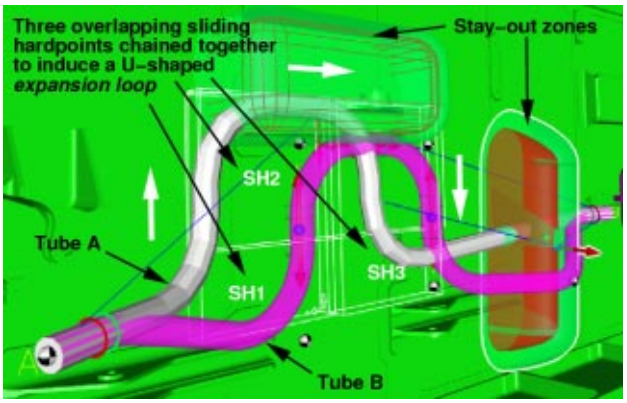

Fig. 16 Expansion loop (A) before and (B) after imposition of stay-out zones

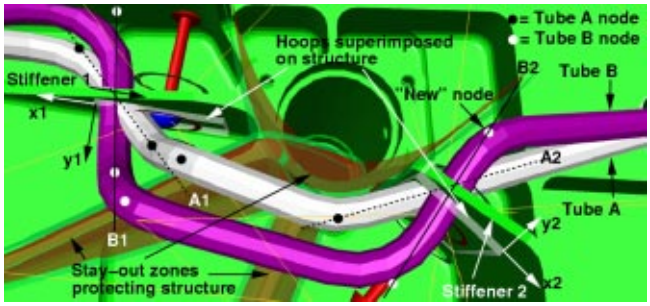

Fig. 17 Tube passing through two stiffening webs (A) with and (B) without equal incidence angles

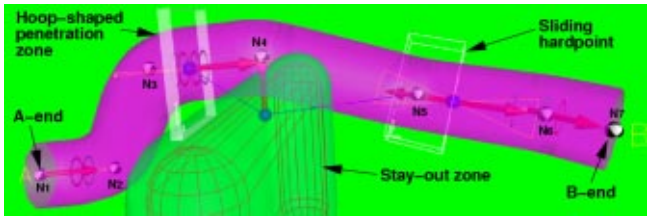

Fig. 18 A timing example (see Table 1)

Table 1 Timing for the case of Fig. 18.

Phase of Algorithm	Number of Nodes	Continuous Optimization Result	Continuous Optimization Time (sec.)
Initial guess	10	N/A	~0
Easy pass	10	Feasible	1.75
Forward	2	Infeasible	0.0028
"	3	Infeasible	0.0026
"	4	Infeasible	0.0025
"	5	Infeasible	0.0026
"	6	Infeasible	0.0026
"	7	Feasible*	0.51
"	8	Insignif. improvement	1.43
Backward	6	Infeasible	0.0051

*Best result. (All other backward-pass phases infeasible, total backward-pass time=0.97 s; total run time=4.67 s)

constraints with variable transformations); (2) There is a considerable amount of sparseness (or locality) inherent in the problem formulation, which we are not yet exploiting. For example, it should be possible to ''factor'' a large tube into a *series* of nearly independent sub-tubes. (Recall from §2.2, for example, that the interactions across hardpoints are weak.) These sub-tubes would later be re-assembled and subjected to a final optimization to obtain a globally optimal solution; (3) It should be possible to add regularizing, or symmetry-breaking, components to the objective function, to improve the conditioning (and thus the convergence speed) of the more difficult continuous sub-problems; (4) It is possible in principle to obtain all derivatives analytically, rather than resorting to finite differences as we currently do; (5) There are distinct opportunities for parallelization; in particular, although the phases described in §2.5 are largely sequential, we have experimented with parallelizable adaptations of them, with promising results.

Notice that a common *modus operandi* is to re-use a desirable node distribution as a discrete base for exploring continuous trade-offs (i.e., changing the soft-constraint weights without recalculating the node distribution). This usually takes only a few seconds per trade-off, even with the current un-optimized code.

4 Discussion and Future Work

As pointed out in §1.1, one class of nonholonomic motion planners for mobile robotics uses a subdivision approach to transform a holonomic ''guess'' into a feasible nonholonomic path. We observe that our algorithm shares this trait in spirit, but with an important difference: rather than trying to fit a nonholonomic section *locally*, i.e., on each side of a breakpoint, we simply use the breakpoint as a guidepost for re-parameterization, and optimize a nonholonomic path *globally* at each subdivision. This means that the nonholonomic solution for a given subdivision might look very different from the solution at subsequent subdivision.

Many researchers (e.g., [14,32–34]) have studied optimal paths subject to *curvature* constraints. This is very different from a constraint on the bend angle *per se*, for, although nonholonomic planners *tend* to produce paths that obey MINANG for some unspecified value, with tubing it *must* be obeyed. Tubing is the only domain we are aware of that exhibits this unusual property.

Targets, i.e., constraint objects that a tube *must* penetrate (generally with a constrained orientation), are a double-edged sword. On the one hand they provide an intuitive ''visual language'' with which a designer can express the gross route of a tube, circumventing the need for a high-level ''router.'' The human designer serves (and must serve) this role by laying out the targets unambiguously. On the other hand targets bring the node-distribution problem into full relief: In our experience, optimal, non-orthogonal routes that account for the MINANG, MAXANG, and MINLEN constraints are relatively easily obtained when targets are not present, i.e., when the only extrinsic constraints are obstacles. In particular, for a given number of nodes a good spatial distribution of them often follows readily from a continuous optimization process. But with targets, nodes generally cannot migrate through space freely, so the problem becomes inherently discrete. This issue is what motivated the development of the AUTOOPT package. (Notice that [9], lacking targets, treats the *number* of bends explicitly, but not their *discrete* distribution relative to the environment.)

In §2.3 we mentioned mixed-integer methods as a way of attacking what is essentially a mixed-variable problem. Audet and Dennis [23] treat mixed-variable programming more directly, and there is a close relationship between [23] and our work: essentially, we use a specially adapted version of their *search* phase, tailored to exploit the geometry of our problem domain. However, at each discrete configuration we solve the continuous sub-problem *completely*, whereas [23] admits the possibility of abandoning a given discrete configuration because progress there is too slow.

We observe that our method, when viewed as a mixed-variable optimization process, takes a heuristic approach, essentially *pruning* many discrete configurations based on high-level geometric considerations (§2.5). Empirically, the heuristics we use seem to be very effective at selecting node distributions that are visually appealing to human designers. Such aesthetic considerations are more important in real design scenarios than one might suspect.

Tube routes may be viewed as optimal *trajectories*, as for spacecraft. In this respect, our method shares some of the flavor of the transcription approach described in Betts [35,36]. Although the constraints that are peculiar to tubes do not normally show up in flight trajectories, nevertheless the methods of [35] could probably be applied to our continuous sub-problems. We note that a rocket moving with a constant speed, subject to constant-magnitude bursts from a sideways-pointing engine, with finite coasting periods in between, will follow a path similar to a tube's. The only treatment we are aware of for this scenario is [37], which deals with the trivial case of one arc followed by one straight section.

For a given continuous sub-problem, one could regard the associated linkage (§1.2) as an actual physical mechanism. Cost functions would represent spring energies, and the exact constraints would represent joint limits. The goal would be to numerically simulate the settling of the mechanism to equilibrium. Several well-known commercial products could be brought to bear on this (e.g., DADS and ADAMS). We have not pursued this program, because only the equilibrium state is of interest (reaching it via a physically realistic trajectory is not advantageous).

Future Work. Future work will treat, among other subjects, (1) the performance improvements discussed in §3; (2) incorporation of other continuous optimizers besides NPSOL (we have experimented with Boeing's own OPTLIB package [38,39] but we do not yet have sufficient data to report results); (3) refinement of the graphical user interface and porting it to a popular system like CATIA or Pro/ENGINEER; (4) the problem of routing bundles or clusters of tubes simultaneously (we have started work on this problem but do not have results to report yet); (5) extension to other transport element types such as wires, hoses, and cables; (6) use of robot-path-planning methods such as in [3,13] to produce a better initial guess;[3] and (7) research on data-management problems such as the robust association of constraint objects to background geometry, and managing the order in which tubes are generated. Ultimately, it should be possible to store only constraint objects, with centerlines being regenerated as needed.

We also plan to investigate the application of the general principles and software architecture we have developed to new areas of automated design. Our preliminary investigations have focused on welded-duct jig layout, assembly tool design, and the design of brackets of various kinds.

Another area that we plan to investigate is the use of more

[3]This idea was suggested by J.-P. Laumond in a personal communication.

versatile, subtly shaped stay-out zones. We are currently experimenting with *adaptively sampled distance fields* (ADFs) [40] for this purpose. An ADF is essentially a look-up table and interpolation scheme for rapidly computing the distance from any point to the surface of an object. Thus, rather than ''shielding'' background objects with a pre-defined library of shapes (e.g., the rounded boxes in Fig. 15), we would instead build an ADF around them. Preliminary results suggest that this method can provide a much better representation of the background shape, while still permitting fast computation.

5 Conclusions

We have introduced a transport-element design method based on an arsenal of *constraint objects* that capture the key aspects of transport element design, and have demonstrated their effectiveness in the domain of metal tubes. Our constraint-based approach allows a designer to work at a high level of abstraction, and permits trade-offs to be made directly in the engineering parameters of interest. These advances speed up the design process, and produce designs that are superior in terms of performance (e.g., lower weight). They can also help to streamline business processes by ensuring that manufacturing and other constraints are consistently satisfied early on, thus eliminating expensive rework. The method dispenses with the view of tubes as fixed linkages that must be redesigned by a human in order to supply enough degrees of freedom to accommodate changes in the surrounding structures. This opens up the possibility of automatic re-design when the background geometry changes, effectively enabling a new degree of automated collaboration between different design domains.

Acknowledgments

The author is grateful to the reviewers for their helpful comments and suggestions.

This work originated through the *KIRTS/Genesis* project, a Boeing systems-design effort led by Dr. Jeff Heisserman and based on his Generative Design research, and extended at Boeing in collaboration with Dr. Sean Callahan.

The author benefited from many discussions with Drs. Heisserman and Callahan, and Drs. Raju Mattikalli and Thomas Grandine. The idea of applying NPSOL is due to Dr. Grandine. Eric Haberman first described many user requirements. Mary Hopwood and Matt McMullen provided critical feedback. Drs. John Dennis, John Betts, Paul Frank, and Evin Cramer, Sharon Filipowski, Alan Jones, Jan Vandenbrande, Yu-Feng Wei, Jean-Paul Laumond, Jean-Claude Latombe, and Lydia Kavraki kindly provided advice and feedback.

Dr. Charles Erignac designed and implemented the GUI, and contributed invaluable architectural and programming expertise. Dr. Dwight Fujimoto contributed key mathematical code. Greg Green and Bob Perry contributed software help.

This project would not have been possible without the vision, support, and drive of Mark Williams of the Boeing Company.

References

[1] Jain, D., Unemori, A., Chatterjee, M., and Thangam, N., 1992, ''A Knowledge-Based Automatic Pipe Routing System,'' ASME Comput. Eng., **1**, pp. 127–132.

[2] Mitsuta, T., Wada, Y., Kobayashi, Y., and Kiguchi, T., 1986, ''A Knowledge-Based Approach to Routing Problems in Industrial Plant Design,'' In *6th International Workshop on Expert Systems and Applications*, pp. 237–255, Avignon, France.

[3] Zhu, D., and Latombe, J.-C., 1991, ''Mechanization of Spatial Reasoning for Automatic Pipe Layout Design,'' AI EDAM 5(1), pp. 1–20.

[4] Zhu, D., and Latombe, J. C., 1991, Pipe Routing=Path Planning (With Many Constraints). In *Proc. IEEE Intl. Conf. Robotics and Automation*, Sacramento, CA.

[5] Bohle, D., Jakobs, G., Hänisch, K., and Kalbitz, H., 1982, ''Rechnerunterstützung bei der Rohrleitungs-Konstruktion im 3D-Raum. (Computer-assisted pipe routing in 3-D space.)'' Chem.-Ing.-Tech., **54**(3), pp. 241–246. (In German.)

[6] Satyanarayana, M. V. V., Rao, A. A., et al. 1992, ''An Automated Pipe-Route Planner in Three Dimensional Plant Layout Design,'' *Proc. CompEuro '92, Computer Systems and Software Engineering*, IEEE-0-8186-2760-3/92.

[7] Wangdahl, G. E., Pollock, S. M., and Woodward, J. B., 1974, ''Minimum-trajectory Pipe Routing,'' J. Ship Res., **18**(1), pp. 46–49, March.

[8] OptiPlant, 2002, by ASD Global, Inc. (USA). http://www.asdglobal.com.

[9] Szykman, S., and Cagan, J., 1996, ''Synthesis of Optimal Non-Orthogonal Routes,'' ASME J. Mech. Des., **118**(3), pp. 419–424.

[10] Conru, A. B., and Cutkosky, M. R., 1993, ''Computational Support for Interactive Cable Harness Routing and Design,'' ASME DE-Vol. 65-1, Adv. in Design Automation, Vol. 1, pp. 551–558.

[11] Conru, A. B., 1994, ''A Genetic Approach to the Cable Harness Routing Problem,'' In *Proc. First IEEE Conf. on Evolutionary Comp.*, IEEE 0-7803-1899-4/94, pp. 200–205, Orlando, FL.

[12] Latombe, Jean-Claude. *Robot Motion Planning*. Kluwer International Series in Engineering and Comp. Sci., 124, 1990.

[13] J.-P. Laumond (Ed.), 1998, *Robot Motion Planning and Control*. Previously published as ''Lecture Notes in Control and Information Sciences 229,'' Springer, SIBN 3-540-76219-1, 343p. Out of print but available free of charge at http://www.laas.fr/jpl/book-toc.html.

[14] Sussman, H. J., 1995, ''Shortest Three-dimensional Paths with a Prescribed Curvature Bound,'' In *Proc. 34th IEEE Conf. Decision and Control*, New Orleans, LA, Dec.

[15] Laumond, J.-P., Jacobs, P. E., Taix, M., and Murray, R. M., 1994, ''A Motion Planner for Nonholonomic Mobile Robots,'' IEEE Trans. Rob. Autom., **10**(5), pp. 577–593, October.

[16] Quinlan, S., and Khatib, O., 1993, ''Elastic bands: Connecting path planning and control,'' *Proc. IEEE Intl. Conf. Robotics and Automation*.

[17] Wei, Y. F., and Thornton, A. C., 2000, ''Tube Production and Assembly Systems: The Impact of Compliance and Variability on Yield,'' In *Proc. ASME Des. Automation Conf*, DETC2000/DAC-14271, Sep.

[18] TubeExpert Corp. (Ger.) http://www.tubeexpert.com, 2002.

[19] Eagle Precision Technology, Inc. (USA) http:/www.eaglept.com, 2002.

[20] CATIA by Dassault Systemes (France) http://www.dsweb.com, 2002.

[21] XpresRoute and PathXpres by Solid Edge, Inc. (USA) http://www.solid-edge.com, 2002.

[22] Das, I., and Dennis, J. E., 1998, ''Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems,'' SIAM J. Optim., **8**(3), pp. 631–657.

[23] Audet, C., and Dennis J. E., Jr., 1999, ''Pattern Search Algorithms for Mixed-variable Programming,'' Technical Report TR99-02, Dept. of Comp. and App. Math., Rice University.

[24] Goux, J. P., and Leyffer, S., 2001, Solving Large MINLPs on Computational Grids. Tech. Report NA/200, U. Dundee.

[25] Duda, R. O., and Hart, P. E., 1973, *Pattern Classification and Scene Analysis*. Wiley, New York.

[26] Friedman, J. H., 1990, ''Multivariate Adaptive Regression Splines,'' PUB 4960, Tech. Rep. 102, Stanford Linear Accelerator Center, Stanford University.

[27] Orr, M., Takezawa, K., Hallam, J. et al. 2000, ''Combining Regression Trees and Radial Basis Function Networks,'' Int. J. Neural Syst., **10**(6), pp. 453–465.

[28] Stanford Business Software, Inc. (USA) http://www.sbsi-sol-optimize.com, 2002.

[29] Gill, P. E., Murray, W., and Wright, M. H., 1981, *Practical Optimization*. Academic Press, London.

[30] Concert Technology by ILOG, Inc. (USA) http://www.ilog.com, 2002.

[31] EZMod by Modellium Corp. (France) http://www.modellium.com, 2002.

[32] Agarwal, P. K., and Wang, H., 2001, ''Approximation Algorithms for Curvature-constrained Shortest Paths,'' SIAM J. Comput., **30**(6), pp. 1739–1772.

[33] Dubins, L. E., 1957, ''On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents,'' Am. J. Math., **79**, pp. 497–516.

[34] Venditelli, M., Laumond, J. P., and Nissoux, C., 1999, ''Obstacle Distance for Car-like Robots,'' IEEE Trans. Rob. Autom., **15**(4), pp. 678–691.

[35] Betts, J. T., and Huffman, W. P., 1993, ''Path-Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming,'' J. Guid. Control Dyn., **16**(1), pp. 59–68, January-February.

[36] Betts, J. T., 1998, ''Survey of Numerical Methods for Trajectory Optimization,'' AIAA J., Guidance, Control, Dynam. **21**(2), pp. 193–207, March-April.

[37] Shald, S. M., 1997, ''Arcline Guidance,'' Master's thesis, University of Minnesota.

[38] Betts, John, and Frank, Paul, OPTLIB and SOCS Math & Information Software, the Boeing Company, http://www.boeing.com/phantom/math_infosw, 2002.

[39] Betts, J. T., and Frank, P. D., 2001, ''OPTLIB: Optimization and Optimal Control Software Library,'' Boeing Information and Support Services—Mathematics and Computing Technology Report M&CT-TECH-01-014, The Boeing Company, P.O. Box 3707, Seattle, WA 98124-2207.

[40] Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R., 2000, ''Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics,'' Mitsubishi Electric Research Laboratory, TR2002-15, Cambridge, MA, April.