SANDROS: A Dynamic Graph Search Algorithm for Motion Planning

Pang C. Chen and Yong K. Hwang, *Member, IEEE*

Abstract—**We present a general search strategy called SANDROS for motion planning, and its applications to motion planning for three types of robots:**

1) **manipulator;**
2) **rigid object;**
3) **multiple rigid objects.**

SANDROS is a dynamic-graph search algorithm, and can be described as a hierarchical, nonuniform-multiresolution, and best-first search to find a heuristically short motion in the configuration space. The SANDROS planner is resolution complete, and its computation time is commensurate with the problem difficulty measured empirically by the solution-path complexity. For many realistic problems involving a manipulator or a rigid object with six degrees of freedom, its computation times are under 1 min for easy problems involving wide free space, and several minutes for relatively hard problems.

Index Terms—**Collision avoidance, configuration space, motion planning, path planning.**

I. Introduction

THE motion planning problem is the problem of finding a collision-free motion for one or more moving objects between the start and the goal configurations. A moving object can be a rigid object, or a jointed object such as an industrial manipulator. Motion planning has received much attention for the past two decades from robotics in order to

1) automatically generate the movements of mobile robots and their arms;
2) automatically plan and program the motions of manufacturing robots and mechanical parts in assembling products.

Automation of motion planning offers a number of advantages over the existing alternatives. It relieves human workers of the continual burden of detailed motion design and collision avoidance, and allows them to concentrate on the robotic tasks at a supervisory level. Robots with an automatic motion planner can accomplish tasks with fewer, higher-level operative commands. Robotic tele-operations can then be made much more efficient, as commands can be given to robots at a coarser time interval. With an automatic motion planner and appropriate sensing systems, robots can adapt quickly to unexpected changes in the environment and be tolerant to modeling errors of the work space. In manufacturing applications, an automatic motion planner can be used to verify the assemblability of a product by finding the motion of each part during the assembly, and to plan the motions of robot manipulators performing operations such as cutting, welding, and assembling. This reduces the product design time by enabling product designers to make necessary modifications at the design stage, and it also reduces the setup time of manufacturing lines by enabling rapid programming of manipulator motions.

Although humans have a superb capability to plan motions of their body and limbs effortlessly, the motion planning problem turns out to be a high complexity problem [47]. The best known algorithm [9] has a complexity that is exponential in the number of degrees of freedom (DoF), and polynomial in the geometric complexities of the robot and obstacles in its environment. For motion planning problems in the three-dimensional (3-D) space where $\mathrm{DoF} = 6$, existing complete algorithms [6], [40] that guarantee a solution often takes tens of minutes of computation time. On the other hand, fast heuristic algorithms [27] may fail to find a solution even if there is one.

Because of the proven complexity of motion planning, the development of a motion planner capable of solving all problems in a short time is unlikely. Hence, the next best alternative is to have an algorithm with performance commensurate with task difficulty. (Because there is no set of benchmark problems, we measure the task difficulty empirically by the complexity of the solution path, e.g., the number of monotone segments comprising the path and the clearance to obstacles along the path.) In this paper, we present an efficient, resolution-complete search framework called SANDROS that results in motion planners with exactly this characteristic. It can be used for motion planning of a serial-linkage manipulator, a rigid object or multiple rigid objects, and can solve "easy" problems quickly and "difficult" problems systematically with a gradual increase of computation time. For typical problems involving 6-DoF robots, the running time ranges from under 1 min to 10 min.

This paper summarizes our work reported in [12], [26], and [29]. Previous work is reviewed in Section II. In Section III, the SANDROS search strategy is described and used to develop motion planners for a manipulator, a rigid object and multiple rigid objects. Their performance is illustrated with examples in Section IV, and conclusions and future work are discussed in Section V. Before we go on, we define some terms here. The *robot* refers to the moving object, and the

Manuscript received January 16, 1996; revised January 8, 1997. This work was supported by the U.S. Department of Energy under Contract DE-AC04-94AL85000. This paper was recommended for publication by Associate Editor M. Peshkin and Editor S. Salcudean upon evaluation of the reviewers' comments.

The authors are with the Intelligent Systems and Robotics Center, Sandia National Laboratories, Albuquerque, NM 87185 USA.

Publisher Item Identifier S 1042-296X(98)02665-2.

configuration vector of a robot is a vector of real-valued parameters that completely determines the position of every point of the robot. For a manipulator, the set of joint angles can be used to define its configuration, whereas for a rigid object the position of a reference point of the object and its orientation about that point serve as its configuration vector. The length of the configuration vector is defined as the DoF of the robot. The configuration space (C-space) is the space of all possible configurations of the robot. We use the terms "free," "collision-free," and "feasible" interchangeably. Since our motion planners use only the distance information between the robot and obstacles, they are independent of the object representation scheme, as long as the distance can be computed.

II. Previous Work

Comprehensive reviews of the work on motion planning can be found in [35], [28]. In this section, we concentrate on major work on manipulators and rigid objects. Planning motions for a general 6-DoF robot is very difficult, and the most efficient motion planners that are guaranteed to find a solution if one exists run in tens of minutes for difficult problems. It should be noted that the computation times for 6-DoF robots are important, because they are useful for general purpose manipulation and in wide use for industrial applications such as spot-welding and painting. (A 6-DoF robot is required to arbitrarily position and orient an object within the work space.) There are several complexity results for various motion planning problems. In [47], planning the motion of a polyhedral manipulator among polyhedral obstacles is proven to be PSPACE-hard. The problem of translating multiple rectangles are shown to be PSPACE-hard in [24], and PSPACE-complete in [25]. An upper bound for path planning is established in [49], which gave an exact algorithm that is polynomial in the geometric description of the robot and environment, and doubly exponential in DoF. The doubly exponential component is later improved to a single exponential with the *roadmap* algorithm in [9]. To sum up, motion planning algorithms are likely to be exponential in DoF.

Motion planners can be classified into two categories:

1) complete;
2) heuristic.

Complete motion planners can potentially require long computation times but can either find a solution if there is one, or prove that there is none. Heuristic motion planners are fast (typically take seconds), but they often fail to find a solution even if there is one. Presently, all implementations of complete algorithms are resolution-complete, meaning they guarantee a solution at the resolution of the discretization used in the algorithm. For this reason, we use the term *complete* algorithm to mean *resolution-complete* algorithm for the rest of this paper. Motion planning approaches are reviewed first, followed by motion planners specifically for a manipulator, a rigid object and multiple rigid objects, respectively. For a fair comparison of planners' efficiency, computation times are calibrated to estimated running times on a modern 200 MHz workstation.

A. Motion Planning Approaches

To date, motion planning approaches can be classified into four categories [35]:

1) skeleton;
2) cell decomposition;
3) potential field;
4) subgoal graph.

In the skeleton approach, the free space is represented by a network of one-dimensional (1-D) paths called a *skeleton*, and the solution is found by first moving the robot onto a point on the skeleton from the start configuration and from the goal, and connecting the two points via paths on the skeleton. This approach is intuitive for two-dimensional (2-D) problems, but becomes harder to implement for higher-DoF problems. Algorithms based on the visibility graph [39], the Voronoi diagram [44], and the *silhouette* (projection of obstacle boundaries) [9] are examples of the skeleton approach. In the cell-decomposition approach [45], [51] the free space is represented as a union of cells, and a sequence of cells comprises a solution path. For efficiency, hierarchical trees, e.g, octree, are often used. In the potential-field approach, a scalar potential function that has high values near obstacles and the global minimum at the goal is constructed, and the robot moves in the direction of the negative gradient of the potential. In the subgoal-graph approach, subgoals represent *key* configurations expected to be useful for finding collision-free paths. A graph of subgoals is generated and maintained by a global planner, and a simple local planner is used to determine the reachability among subgoals. This two-level planning approach has turned out to be the most effective path planning method, and is first reported in [18].

B. Manipulator Problems

The first complete planner for a 3-DoF manipulator is developed using an octree representation of the C-space obstacles in [17], while sequential construction of the C-space obstacles is used in [40] to plan motions for a general serial manipulator in the 3-D world space. The latter computes the free regions of the C-space by successively computing feasible ranges for each link, starting from the base of the robot. For example, for each point in the feasible ranges of $joint_i$, the feasible ranges of $joint_{i+1}$ are computed. These ranges are grouped into regions, and the search for a path is done on the graph of these regions. The number of feasible ranges are exponential in DoF, and this algorithm is likely to take a few minutes in our estimation for a 4-DoF problem.

Kondo [34] has reported a fast grid search algorithm that uses several heuristics to favor different motions during search. The effectiveness of a heuristic is measured by $l^{\mathrm{DoF}}/N_{\mathrm{coll}}$, where l is the length of the path from the start to the current search configuration, and N_{coll} is the number of the collision checks performed by the heuristic to generate the path. The numbers of collision detections reported by this planner are roughly on the same order of magnitude as those of our planner. Kondo's algorithm is effective when the solution path lies in a narrow winding tunnel because of the grid search, but can spend a lot of computation exploring a large dead-end space. This algorithm is also used to plan motion of a rigid body.

Probabilistically complete algorithms find a solution with probability 1 given long enough computation time, and is a typical characteristic of randomized algorithms. In [4] a potential-field based method called RPP uses gradient descent to approach the goal configuration. If it gets stuck at a local minimum, the planner uses random walks to escape from the local minimum. This process is repeated until the robot reaches the goal. A more elaborate algorithm, also potential based, applying a dynamic programming technique to various submanifolds of the C-space is presented in [3]. It is slower but capable of solving harder problems than RPP. The random nature of RPP prevents one from devising an unsolvable problem, but when the solution path has to pass through a narrow space, it takes a long time to find this passage with random movements. A subgoal graph called *probabilistic roadmap* is developed in [31] to overcome this problem. Random configurations are generated and those without collisions are kept in the graph. A simple local planner is used to check the reachability between subgoals. The graph is augmented with additional subgoals in the cluttered C-space to increase the connectivity of the graph. Finally, path planning is done by connecting the start and goal to subgoals on the graph and traversing the graph to connect the two subgoals. The main difference between this and our algorithm is that our algorithm generates subgoals hierarchically and incrementally until the problem is solved, while theirs generates all the subgoals at the beginning. When solving multiple problems in the same environment, the entire subgoal graph can be generated before solving the problems, or a learning technique can be used to progressively build the graph while solving the problems one by one [11]. Both of the probabilistic algorithms can also be used for rigid objects.

Paden *et al*. [45] have presented a complete algorithm which computes the free C-space in the 2^{DoF}-tree of rectangular cells using the Jacobian of the robot. Although developed for a manipulator, generalization to a rigid body is straight forward. The bound on the Jacobian gives the maximum distance traveled by any point on the robot when the robot configuration is varied inside a unit sphere. Thus, the distance between the robot and objects divided by the bound on the Jacobian gives a lower bound on the radius of the collision-free sphere in the C-space. For a manipulator with a long reach or an elongated object, however, the large norms of Jacobians allow only a small volume of the free C-space to be computed at a time. It takes about 1 s to solve a 2-DoF problem.

There are many heuristic motion planners with computation times proportional to their capabilities. For example, the algorithm in [46] uses a global planner based on the Voronoi diagram and a local planner based on the virtual spring method (a potential field variant), and its computation times are under 1 min for an easy 6-DoF problem in 3-D. On the other hand, the algorithm in [22], [23] is much more elaborate, and uses a sequential framework to decompose the original DoF-dimensional problem into a sequence of smaller-dimensional problems. Given the path of the ith joint parameterized by t, it plans the path of the $(i+1)$th joint in the $t \times \theta_i$ space using a visibility or potential-field method. This algorithm solves two problems involving a 7-DoF robot in relatively cluttered environments in 10 min and 1 h. For many applications, a specialized heuristic motion planner is often sufficient, and a computationally expensive complete planner may not be needed. We note that our manipulator motion planner exploits the sequential structure of a manipulator as done in [22] and [40].

C. The Classical Mover's Problem

There has been a lot of work on the classical mover's problem, i.e., motion planning for a rigid object. For the 2-D problem (3 DoF) an octree is used in [7] to develop a complete planner, while for the 3-D problem (6 DoF) the first implemented planner is a grid search algorithm [14]. It builds up a lattice of points in the C-space, and collision check for a point is done by evaluating the boundary equations of the C-space obstacles. It uses several heuristics to speed up the search. Its computation time is estimated to be tens of minutes for hard problems. In [2], an exact motion planner for a polygonal robot with 3 DoF is developed. This algorithm builds an exact geometric description of the C-space obstacles using edge-edge contact conditions.

A subgoal-graph approach is used in [20] to plan motions of a polygon in 2-D. Random subgoals are generated during the search, while a local planner is used to connect the subgoals. It solves 2-D, 3-DoF problems under 1 min, and is one of the first randomized path planners. Another subgoal-graph approach, reported in [6], uses a genetic algorithm to select subgoals and search collision-free paths between subgoals. It is a complete planner with run times in tens of minutes for difficult 6-DoF problems. For other complete rigid-body motion planners, see also the previous section.

In [27], a fast heuristic planner for 3-D is developed based on the potential field. It finds a path for the reference point of the robot first, and moves robot's reference point along this path while aligning robot's longest axis tangentially to the path to minimize the swept volume. If a collision occurs, it generates feasible robot orientations in that region, and uses these as intermediate subgoals. This algorithm is fast and effective for convex robots, but fails on problems with highly concave robots like that in Fig. 9.

For a 2-D rigid object the C-space is only 3-D, and a brute force search algorithm runs in a few minutes. A brute force search would discretize the C-space, check whether there is a collision at each of these points, and find a connected sequence of collision-free points in the C-space. In fact, such an algorithm has been implemented in [37]. For the 3-D classical mover's problems, however, a more efficient algorithm is still needed to search 6-DoF space.

D. The Multimovers' Problem

The multimovers' problem is the problem of finding paths for several robots between their initial and goal configurations. Robots have to avoid one another as well as obstacles. The algorithms for this problem can be classified into centralized and distributed planning. In centralized planning, motions of all robots are planned by a single decision maker. Centralized planning can be further divided into algorithms with or without priority among the robots. In certain applications, there are priority relations among the relative importance of the missions of robots. In other cases, the priority is artificially imposed to decompose the multiple movers' problem into a sequence of single-mover's problems. This, however, introduces a loss of optimality as well as completeness, i.e., the planned motion of a robot may preclude the motion of another robot with a lower priority.

A centralized approach is reported in [16], where the obstacles in the composite C-space are constructed to plan paths for two robots. In case there is a conflict between robots, the burden of collision avoidance is imposed on the robot with a lower priority. Motion planning of many square-shaped robots in the plane is presented in [8]. This algorithm has an application in an assembly cell, where multiple robots move on the ceiling, delivering parts and performing assembly operations. This algorithm tries to move as many robots as possible in straight lines, and minimizes the number of robots that have to make turns. Recently, game theory is used in [36] to plan paths of robots while optimizing multiple objectives. Robots are constrained to move on a skeleton such as the Voronoi diagram, and the concept of Nash equilibrium is used to schedule robot motions on the skeleton. A randomized approach is used to plan motions of multiple manipulators in [33]. This algorithm plans manipulator motions that move an object from one position to another. The path of the object is planned first, and the motion of each manipulator that moves the object along the planned path is computed by solving inverse kinematics. Since manipulators cannot carry the object outside their reachable space, the planner assigns specific manipulators to carry the object at different portions of the path. Although this algorithm is not complete, it is efficient and solves many realistic problems. Finally, RPP and the probabilistic roadmap are used in the composite C-space to plan motions of multiple robots in [5] and [48], respectively.

A distributed planning approach is taken in [53] to plan paths for circular robots. Each robot plans its own path using the visibility graph. When conflicts arise among robots, it is reported to the central coordinator called the *blackboard,* which issues priorities based on the tasks and current state. Another decentralized approach for circular robots of different sizes is presented in [38]. A quadtree is used for path planning along with a Petri net formulation to resolve the conflicts among robots.

III. Algorithm

We have developed an efficient, resolution-complete algorithm capable of planning motions for arbitrary robots operating in cluttered environments. We plan robot motions in the C-space, and make the standard assumption that if a task is solvable, a solution path can be represented by a sequence of unit movements in the C-space, discretized to a user-specified resolution. To handle the high dimensionality of the C-space and search complexity, we make the following design decisions without sacrificing completeness.

1) First, we determine whether a given point in the C-space is in free space by computing the distance [19] between the polyhedral robot and obstacles. (As long as a CAD modeler supports a distance routine, it can be used with our motion planner.) Contact conditions can be used [14], [40], but equations describing contact boundaries are nonlinear, and detecting intersections of these boundaries is computationally expensive. The use of distance makes our algorithm independent of object models, and enables the selection of 'safer', i.e., larger clearance, configurations for the robot.
2) Second, we use a two-level hierarchical planning scheme to reduce memory requirements, as done in [18]. It is difficult to store all the collision-free points even if we could compute them all, since the C-space typically has an enormous number of points even at a coarse resolution. We circumvent this problem by planning at two levels using a global and a local planner. The global planner keeps track of reachable, unreachable, and potentially reachable portions of the C-space, and the local planner checks the reachability of a portion of the space from a point. If a portion of space is reachable from a point, then the corresponding collision-free motion needs not be stored, since they can be readily recovered by the local planner at any later stage of the algorithm. But if the distance computation is expensive, it is more efficient to store the motion.
3) Third, we use a multiresolution approach to reduce search time. An exhaustive search for a collision-free motion is prohibitive because of the enormous size of the C-space. Yet, heuristic algorithms that do not examine the entire space are inevitably incomplete. To achieve both time efficiency and completeness, we use the global planning module to first search promising portions of the C-space at a coarse resolution. It increases the resolution to finer levels only if a solution is not found at the coarse level, and only in promising portions of the C-space. The planner searches the space both heuristically and systematically so that each motion planning problem can be solved in time according to its difficulty.

These design decisions are embodied in a new search strategy called SANDROS, which stands for *Selective And Non-uniformly Delayed Refinement Of Subgoals.* Given two points s and t representing the start and goal configurations of a robot, we maintain a set of subgoals to be used by the robot as guidelines in moving to the goal configuration. Subgoals represent portions of the C-space that have relatively large clearances to obstacles, and hence correspond to configurations that are easy for the robot to reach using the local planner. Initially, we maintain only a small number of 'big' subgoals, each of which represents a large portion of the C-space. Because these subgoals are big, they provide only coarse guidelines for the robot to follow. A collision-free motion can be found very quickly with these coarse guidelines if the problem is easy. If a collision-free path cannot be found with these subgoals, then some of the subgoals are broken down to several smaller, heuristically selected subgoals to provide more specific guidelines. The process of subgoal refinement is delayed as much as possible, and is performed in a nonuniform fashion to minimize the number of nodes.

At the highest level, SANDROS uses a *generate-and-test* strategy to plan motions. It has two main modules: a *global planner* $\mathcal{G}$ that generates a plausible sequence of subgoals to guide the robot, and a *local planner* $\mathcal{L}$ that tests the reachability of each subgoal in the sequence. Planner $\mathcal{L}$ needs to be deterministic but can use pseudorandom number generators to simulate random but reproducible behavior. If $\mathcal{L}$ succeeds in reaching each subgoal through the tested sequence, then a collision-free path is found. If $\mathcal{L}$ fails to reach a subgoal due to collisions, then $\mathcal{G}$ would first try to find another sequence without any subgoal refinement. If no sequence is available, then a subset of the current subgoals would be refined repeatedly according to the SANDROS strategy until either a sequence becomes available, or no further refinement is possible.

It should be noted that $\mathcal{G}$ is completely independent of $\mathcal{L}$, and that there is a trade-off between the simplicity of $\mathcal{L}$ and the guiding effort of $\mathcal{G}$. If the local planner is as complicated as a complete algorithm, then the global planner will never generate any subgoals because the local planner's range of effectiveness encompasses the entire C-space (assuming a solution exists). At the other extreme, if the local planner is a simple and inflexible algorithm like connect-with-straight-line, then the burden of planning rests heavily on the global planner in that it will have to generate many subgoals before a solution can be found. Finding the optimal subdivision of labor between $\mathcal{L}$ and $\mathcal{G}$ is difficult. However, in principle, $\mathcal{L}$ should implement some 'greedy' algorithm with a capability to 'slide' around simple obstacles, so easy problems can be solved without generating too many subgoals. Finally, we remark that the dynamic graph searching framework of SANDROS can be tailored to various kinds of robots by specifying the subgoal representation and refinement strategy in $\mathcal{G}$ and the sliding strategy in $\mathcal{L}$ (see Section III-C).

A. Global Planning

Global planning takes place in three stages:

1) sequence generation;
2) sequence verification;
3) node refinement.

In the sequence generation stage, the global planner $\mathcal{G}$ finds a "good" sequence of subgoals by searching through a dynamic graph G containing configuration points s and t with additional points and nodes representing subgoals. A point represents a single configuration in the C-space, while a node represents a subset.

1) Graph Construction: The points in G are all points reachable from s or t through applications of $\mathcal{L}$. This set of points P is divided into P_s and P_t representing points reachable from s and t, respectively. A point cost is stored indicating the cost of reaching it from s or t. Each node v of G is classified as either reachable (in U) or not-yet-reachable (in V), depending on whether a reachable point p in P has been found with which $\mathcal{L}$ is able to find a collision-free path from p to a point q in v. Although G may be changing, we maintain the invariant that the nodes of G not intersect each other. Thus, each reachable point p has an unique node $A(p)$ of G that contains p. Each point q reachable from another point p has also a back pointer $B(q) = p$ so that a path from s or t to q can be retraced.

There are two types of edge connections in G:

1) (node)-to-(node);
2) (reachable point)-to-(not-yet-reachable node).

Each edge has a cost estimating the actual cost of traversing from a terminal (node or point) to another. We initialize G by setting $P_s = \{s\}$, $P_t = \{t\}$, $U = \{\}$, $V = \{\mathrm{root}\}$, and connecting s to root and root to t with edges. The root denotes the entire C-space. To control the subgoal refinement process, we also maintain a node queue Q, sorted and grouped by refinement level, and initialized to the empty set.

2) Sequence Generation: To generate a plausible sequence, we simply apply a shortest-path graph algorithm [1] on the subgraph of G induced by V and P, with source vertices P_s and sink vertices P_t. We define the cost of a sequence with end points in P and intermediate nodes in V as the sum of the edge costs plus the costs of the end points. Nodes themselves have no cost. This cost serves only as an estimate of the actual length of a solution going through the subgoals. To restrict the number of possible sequences through a subgoal, we adopt the principle that no other way of reaching a subgoal will be considered once it is declared reachable. We implement this principle by associating exactly one point in P with every node in U. Allowing more than one point per node is also possible, but we favor the one-point-per-node rule for its conceptual and implementational simplicity. (We allow a reached node to be reached again at other points when the node is split into smaller nodes through refinement.)

3) Sequence Verification: If the graph algorithm does produce a sequence with end points in P and intermediate nodes in V, then we enter the sequence verification stage. In this stage, we use $\mathcal{L}$ to determine the connectability of the end points through the sequence of nodes. Let $s' \in P_s$ and $t' \in P_t$ be the end points of this sequence. Let $d(q)$ be the minimum Euclidean distance between the robot at configuration q and the obstacles. We begin by choosing the search direction using $d(s')$ and $d(t')$ as a guide: If the $d(t')$ is smaller than $d(s')$, then we will search backward by starting at t'; otherwise, we will search forward by starting at s'. Heuristically, the point chosen (s' or t') should have less clearance from the obstacles, and hence should be extricated first to constrain the search. Let p be the point chosen and p' be the other point. To search forward, we call $\mathcal{L}$ to check the reachability of the first node v from point p; to search backward, we call $\mathcal{L}$ to check the reachability of the last node v from point p. Either way, if any point, say q, of v is reachable from p with $\mathcal{L}$, we would

1) swap v from V to U;
2) insert q into P;
3) store a pointer $A(q) = v$;
4) connect q to the original neighbors of v in V with new edges;
5) store a back pointer $B(q) = p$, so that a path from s or t to q can be retraced.

Then, we would continue the verification stage by checking whether p' and q can be connected. The verification stage ends

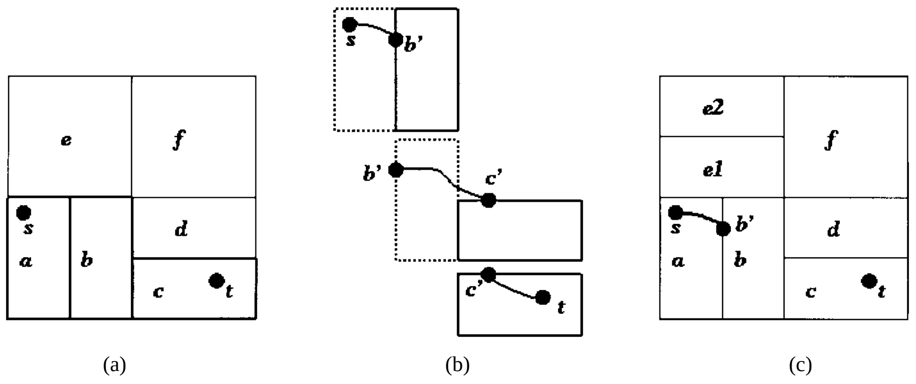


Fig. 1. Sequence verification and node refinement.

when either the entire sequence is connected, or a node v is found unreachable from a point p. In the former, we would retrace a path from s to t through G to yield a motion for the robot. In the latter, however, we would disconnect p from v, push both $A(p)$ and v into Q if they are not in Q already, and return to generating another sequence.

4) Node Refinement: Continuing with the sequence generation process, if the graph algorithm produces no candidate sequence, then we would enter the node refinement stage. In this stage, we modify G continually by refining the subgoals in Q until either a candidate sequence becomes available, or Q becomes empty. We pop off every node v in Q deemed to have the least amount of refinement and refine it into more clear, defined subgoals. After refining v into children $C(v)$, the next step is to modify G to reflect the change in V. We augment G with $C(v)$ by inserting $C(v)$ into V, and connecting every neighbor node of v with every node of $C(v)$ while observing appropriate edge constraints (described in the next section). Also, if $A(B(v))$ has not been refined already, then we would push $A(B(v))$ into Q to ensure the eventual chance that every node of U gets refined.

Fig. 1 illustrates the sequence verification and refinement using rectangular subgoals. Fig. 1(a) shows a refinement of the C-space into subgoals, and a test sequence is shown in bold rectangles. Fig. 1(b) shows $\mathcal{L}$ finding a collision-free path from one subgoal to the next, till the goal is reached (bidirectional search not shown for simplicity). If all sequences fail to generate a solution, and if node $\mathbf{e}$ is chosen for refinement, several new sequences become available for verification by $\mathcal{L}$ [Fig. 1(c)].

B. Node Representation and Refinement

Although subgoal representation is independent of the search framework thus presented, it is nevertheless an important factor in determining the efficiency of the eventual search. For manipulators and rigid bodies, we find the following two approaches to be quite effective.

1) Manipulators: For a manipulator (single open kinematic chain) with n DoF a node at (refinement) level k is a n-vector with only the first k components specified, which is an affine space of dimension $n-k$. Thus, the totally unspecified node v_0 at level 0 represents the entire C-space, and a fully specified node represents a single point. The edge cost between two nodes is defined as the sum of the differences between the coordinates that are specified in both nodes (a modified Manhattan metric). Nodes are connected by an edge only if the edge cost does not exceed a certain threshold T_e.

In the node refinement stage, nodes at the highest level are refined: A node v at level k is refined as follows: First, we use the specified components of v as the first k joint values for the robot. Then, using only the links whose positions are totally specifiable by the first $k+1$ joints, we compute the distance between the links and other objects in the work space for all possible $(k+1)$th joint values at a prescribed resolution called *stride* (see Section III-C). The resulting set of nodes with $k+1$ specified components and positive distance is then filtered into a list of nodes $C(v)$ using a *dominate-and-kill* method. In this method, the process of selecting a node v_c with the maximum distance value, and removing each node whose $(k+1)$th component is within λ number of nodes of v_c is repeated until every node is considered. The idea is to condense the set of possible nodes into a sparse collection of subgoals having maximal clearances based on the specified links. Refinement stops when all nodes are fully specified.

2) Rigid Bodies: For a rigid object, a node v is simply a rectilinear cell of the C-space. That is, for each dimension, a low and high value specify the range of valid configurations. Together, the nodes form a partition of the C-space. Two nodes are connected with an edge if they are adjacent in that their contact area is nonzero. The edge cost between two nodes is defined as the Euclidean distance between the centers of the two cells. In the node refinement stage, nodes with the largest volume, are refined by cutting on the longest side in half. That is, the nodes that contain the most number of lattice points in the discretized C-space are split in half along the dimension containing the most number of lattice points. Refinement stops when the longest side of the nodes contains only one lattice point. For multiple objects, the C-space is the product of the C-spaces of all objects. The form of node, edge and edge cost are similarly defined as for the case of a single rigid object.

C. Local Planning

The local planner simulates robot movements in the C-space in small steps. A step is defined as a configuration change where a DoF is changed by a preset amount called *stride*, which represents the resolution. The stride in each dimension is normalized so that the maximum distance traveled by any point on the robot is about the same for each stride. A point that is within one step of another is a neighbor of that point. Collision checking is done after the robot takes a step. It is possible for the robot to collide with the objects during a step, although it does not before and after the step. The Jacobian method [45] can be used to dynamically control the stride sizes and ensure no inter-step collisions, but we have decided to use small fixed strides for implementation efficiency.

The local planner $\mathcal{L}$ checks the reachability of a node v from a point p by moving the robot from p to any point q in v. If q is found, then the cost of q is recursively defined as the cost of p plus the number of steps needed to move from p to q. The iterative procedure of moving from p toward v is as follows: First, we make progress toward v by considering all neighboring points of p that are one step closer to v than p is, and move to the point p' that has the maximum clearance $d(p') > 0$. (Whether the node is an affine space or a cuboid, the distance between point p and node v is defined as the minimum Euclidean distance between p and v, which is easily computed.) Next, we "slide" repeatedly by considering sequentially in each dimension, the two neighboring points one step away in either direction. If there is a point p'' closer to v than p is, while having a larger clearance $d(p'')$, we would move from p' to p'' and set p' to p''. If no progress can be made, then $\mathcal{L}$ would report a failure; otherwise, the move-toward-and-slide procedure is repeated until v is reached.

D. Post Processing of Path and Optimality

A solution path found typically has many noise-like kinks since $\mathcal{L}$ tries maximize obstacle distance as much as possible while moving toward the goal. To locally optimize the path, we have developed a method called *corner-cut*. It deletes every other points of the solution path, and tests if the robot can move between the remaining points by interpolation. If a collision occurs during interpolated movement between two points, the deleted path point between them is re-inserted to the path. This process of deleting every other point and checking collision via interpolation is repeated until no more point can be deleted. The C-spaces in Fig. 3 show the original wiggly paths and the corner-cut paths consisting of a small number of line segments. A corner-cut path can be further optimized using numerical methods that minimizes a weighted sum of path length, distance to obstacles along the path and other measures [27], [45].

The near-optimality of the path found by our algorithm is supported only qualitatively and empirically. Because SANDROS verifies a node sequence with the smallest cost, a short path is heuristically more likely to be found first than a longer path. Grossly nonoptimal paths are of two types.

1) First, the solution path is found from a sequence of large subgoals. This happens when the free space is wide, but there are enough obstacles to make the robot take a detour. This type of paths can be made near-optimal with respect to homotopy by post-processing in most cases.
2) Second, the solution found and the optimal path are on different sides of an obstacle. The only way to overcome this problem is to continue to search the C-space until a satisfactory solution is found. Note that it is hard to define what is satisfactory and know when to stop searching.

For practicality, one probably has to be content with stopping when the difference of path lengths of two successive solutions becomes small, or no more solution is found within a preset time limit.

E. Completeness Proof

To recapitulate, our algorithm searches for a solution by repeating the process of finding a promising sequence of nodes in a graph G, verifying its feasibility with $\mathcal{L}$, and modifying G with the refinement procedure. The efficiency of our algorithm comes from the fact that we delay the node refinement until there is no candidate sequence available at the current refinement level (G disconnected). Such delay allows us to find a solution quickly in a graph with only a small but representative set of subgoals when intricate maneuvering is not needed.

It is of course possible to refine completely every node of G down to a point first, and then plan a path based on the resulting unique network of points $\bar{G}$. However, it would be terribly inefficient to find the shortest plausible sequence of subgoals because of the sheer size of $\bar{G}$. In fact, for rigid bodies, the resulting network is simply the discretized map of the free C-space, and $\mathcal{L}$ only needs to check whether an adjacent point is collision-free. On the other hand, using a nontrivial refinement parameter λ for manipulators allows us to combine similar subgoals into one single subgoal and utilize the power of $\mathcal{L}$ to connect subgoals that are λ away from each other.

For manipulators, there's no guarantee of finding a solution in a $\bar{G}$ with a nontrivial λ when one exists in the corresponding $\bar{G}$ with $\lambda = 0$. The same lack of guarantee goes for rigid bodies, if we choose not to refine nodes down to the resolution. (Users who fear that there's no solution may wish to do so to reduce the amount of time wasted.) However, for both manipulators and rigid bodies, we can show that if a solution exists in $\bar{G}$, then SANDROS will also find one in G without necessarily refining every node down to a point.

Theorem 1: Suppose that a task of moving from s to t is solvable by first refining the C-space into a network of points $\bar{G}$, and then planning a path through $\bar{G}$ using $\mathcal{L}$ to connect s to t. Then our algorithm is complete in that it can also solve the same problem, but with possibly less node refinement.

Proof: If the problem of moving from s to t is solvable through total refinement, then there must be a loopless sequence $\Gamma_0 = (s = v_0, v_1, \cdots, v_n = t)$ with $v_i \in \bar{G}$ such that $\mathcal{L}$ is able to connect v_i with v_{i+1} for all $i < n$. Suppose that our algorithm fails to solve the same problem, and terminates with a partially refined $G \neq \bar{G}$. Then consider the sequence $\Gamma_1 = (\varphi(v_0), \cdots, \varphi(v_n))$, where $\varphi(v_i)$ denotes the node of G that dominates v_i in that the subspace of $\varphi(v_i)$ contains that of v_i. By contracting any loop of this sequence repeatedly, and then picking the subsequence starting with the last point in P_s and ending with the first point in P_t, we can obtain a loopless (but not unique) sequence of the form $\Gamma_2 = (w_0 \in P_s, w_1, \cdots, w_m \in P_t)$ with $m \leq n$, and each w_i in G. Since Γ_2 contains no repetition of nodes and has a finite cost (because the cost of Γ_0 is finite), it must be a sequence verified by $\mathcal{G}$ to be infeasible. Because Γ_0 is loopless, every fully refined node w_i in Γ_2 must correspond to a unique v_j in Γ_0. Further, for such $i < n$, w_{i+1} must dominate v_{j+1}. Hence, for Γ_2 to be infeasible, there must exist a smallest $k \geq 1$ such that w_k strictly dominates v_k in that w_k is still not fully refined. On the other hand, such k cannot exist for the following reason. If w_k were reachable by the end of our algorithm, then w_k would eventually be pushed into Q. (See the last portion of Section III-A.) If w_k were not reachable, then w_k would have been pushed into Q immediately after this determination. Either way, w_k would have been expanded eventually and totally, and hence cannot strictly dominate v_k. Therefore, our algorithm must have also succeeded by contradiction. □

Thus, we have the guarantee that while minimizing the number of nodes in G, SANDROS' interleaving process of search and refinement will eventually find a solution if there is a solution in the completely refined graph $\bar{G}$.

IV. EXAMPLES

SANDROS search strategy has been applied to motion planning of a manipulator, a rigid object, and multiple rigid objects. SANDROS planner is run on both easy and hard problems, and its performance is illustrated using the following numbers. The computation times are run times on a 200 MHz SGI Indigo2 workstation. The number of nodes on the graph generated by SANDROS, the number of calls to the local planner and the number of distance computations give empirical ideas of the problem difficulty. We used the distance routine in [19] developed for polyhedral objects, and thus robot and obstacle shapes are polyhedral.

To get a meaningful and fair assessment of SANDROS' efficiency, the following measure is developed.

1) First, we use the number of collision detection (or distance) computations between the robot and its environment rather than computation time, since it is independent of geometric complexities of the robot and obstacles.
2) Second, SANDROS' efficiency is compared against a brute-force grid-search method in the C-space, which discretizes the C-space into N_{grid} points, performs collision detection at each point, and finds a path among the collision-free points.

This planner will run in $O(N_{\text{grid}})$ time. Let C_i be the length of the C-space in the ith dimension. Then $N_{\text{grid}} = \prod_{i=1}^{\text{DoF}} C_i/\textit{stride}_i$. The SANDROS' efficiency, $\mathcal{E}$, is then measured by the ratio of N_{dist}, the number of distance computations by SANDROS, to the N_{grid} of the C-space grid. (If we used $N_{\text{grid}} = \prod_{i=1}^{\text{DoF}} C_i/(\lambda * \textit{stride}_i)$, then we would have measured the efficiency of interleaving search with subgoal refinement against performing search after total refinement within SANDROS' framework.)

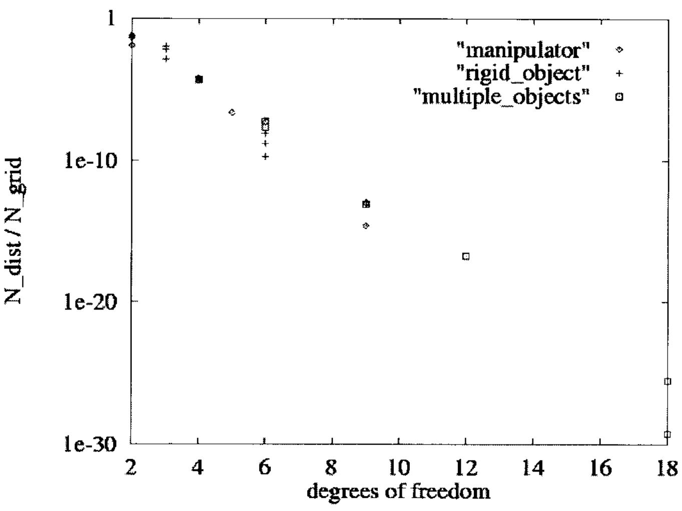


Fig. 2. Ratio of the number of collision-check points to the total number of points in the C-space, plotted in log scale against the degrees of freedom of the robot(s).

In Fig. 2, $\mathcal{E} = N_{\text{dist}}/N_{\text{grid}}$ for the examples that follow are plotted in log scale against DoF, and this empirically shows the efficiency of our algorithm for high-DoF problems. The apparent "log-linearity" exhibited in the plot indicates an exponential time improvement of SANDROS over the trivial planner, even though SANDROS' performance may still be exponential in DoF.

Tables I–III shows detailed statistics on examples: the number of DoF, computation times, number of nodes in the graph, number of local planner invoked, number of subgoal sequences tried, number of distance computations, C-space grid size, search parameter λ, and $\mathcal{E}$. The most important parameters of our planner are the strides, λ and T_e. The strides used are typically 2–5° for rotation, and can be inferred by the grid size in the tables. The stride should be set so that at all points on the solution path the robot should be able to move at least one step along each dimension of the C-space. This will allow the local planner rooms to slide around obstacles. For manipulators, λ denotes the distance between subgoals. The larger it is, the faster the planner runs and the bigger the chance of not finding a solution. The value of ten seems to achieve the most efficiency without causing failures in our examples. The maximum distance between neighbor nodes, T_e = 1.5 λ in all examples. For single or multiple rigid objects λ = 1 is used, but all examples were solved before the lengths of cuboid subgoals reach $\lambda * \textit{stride}$. For practical applications, we recommend using another parameter D_{safe} such that the local planner executes the sliding strategy only when the distance to obstacles is less than D_{safe}, which is set to ∞ in our examples.

We compare the performance of our planner with only those which make point queries to check collision or to compute distance, since an objective comparison is hard to do with different approaches. The average efficiency of our planner measured by the number of distance computations is roughly on the same order of magnitude as those of the planners in [4], [23], and [34]. If we are solving five to ten different problems in the same environment, the planner in [31] is equally efficient. Their performance depends on the problem.

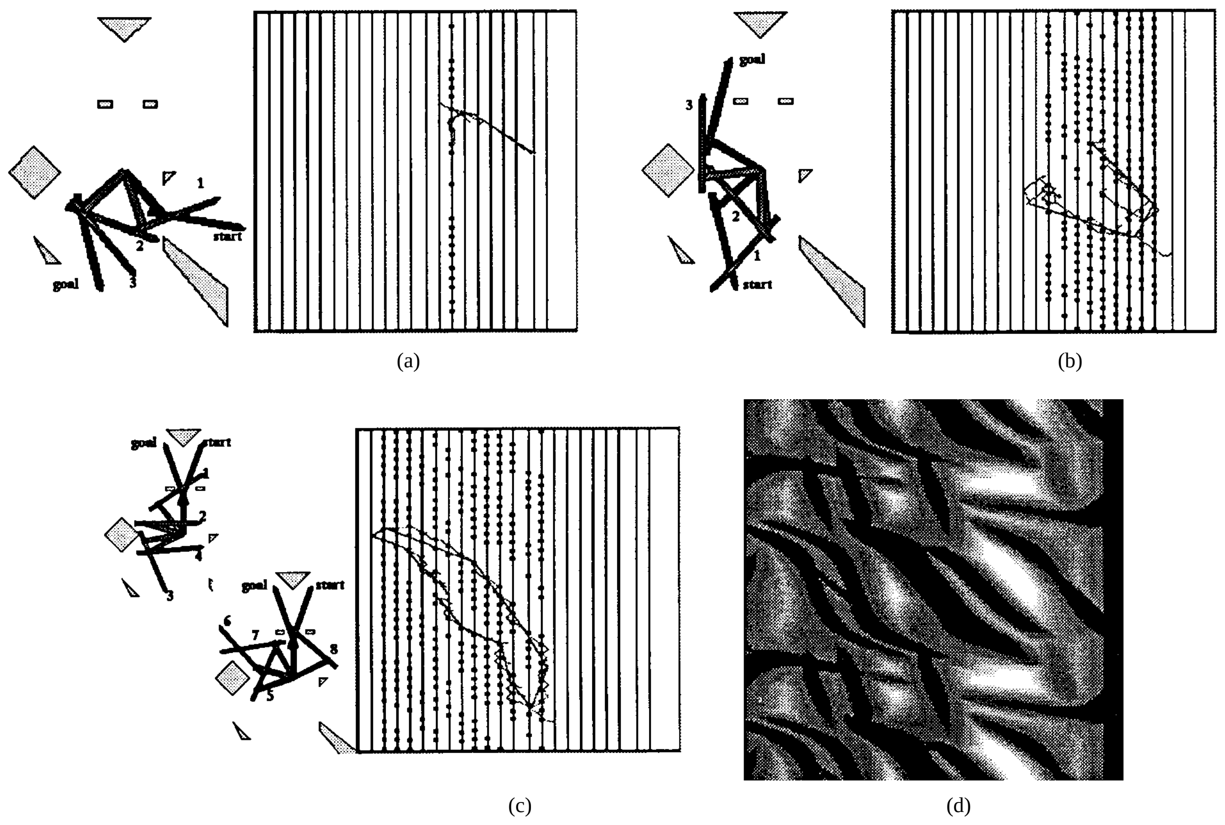


Fig. 3. A planar 2-link manipulator among polygonal obstacles. Three problems with different levels of difficulty are solved in times proportional to the difficulty. The graphs of subgoals in the C-space are also shown. The vertical bars are 1-D subgoals, and the small squares denote zero-dimensional subgoals. The graph sizes reflect the problem difficulties. The curves represent the traces of the local planner, and the dark straight lines are post-processed solution paths.

TABLE I
PERFORMANCE STATISTICS OF SANDROS PLANNER FOR MANIPULATORS

example	dof	time	N_{node}	N_{local}	N_{seq}	N_{dist}	grid size	N_{grid}	λ	$\mathcal{E}$
2link, easy	2	1	45	15	5	1588	300x423	1.27e5	12	1.25e-2
2link, medium	2	2	235	41	14	5469				4.31e-2
2link, hard	2	3	323	97	52	7949				6.23e-2
4link, easy	4	8	266	98	54	10165	170x113	1.45e8	10	4.15e-5
4link, hard	4	14	271	164	90	14986	x113x113			6.12e-5
AdeptOne	5	22	182	94	63	10079	296x171 x42x191x105	4.26e10	10	2.37e-7
Puma260	6	5	46	35	101	2129	155x158x146 x194x81x178	1.00e13	10	5.13e-8
9link, easy	9	10	17	9	2	7842	170x170x170	3.28e18	10	2.49e-15
9link, medium	9	544	1186	714	315	285931	x64x113x64			9.53e-14
9link, hard	9	940	2114	1823	1342	308292	x113x113x113			1.03e-13

N_{node}: no. of subgoals, N_{local}: no. of calls to the local planner, N_{seq}: no. of subgoal sequences tried,

N_{dist}: no. of distance computation, λ: planning parameter, $\mathcal{E}$: efficiency measure = $N_{\text{dist}}/N_{\text{grid}}$.

If the free C-space is a narrow winding tunnel, Kondo's grid search [34] can map out the tunnel with a small number of collision detection checks. Our *local* planner and that in [4] will fail when the tunnel makes a sharp turn, and the planner

TABLE II
PERFORMANCE STATISTICS OF SANDROS PLANNER FOR RIGID OBJECTS

example	dof	time	N_{node}	N_{local}	N_{seq}	N_{dist}	grid size	N_{grid}	λ	$\mathcal{E}$
L shape, 2D	3	1	11	6	5	4736	94x94x80	7.07e5	1	6.70e-3
clip-1nail	3	1	9	8	4	4516	120^2x233	3.36e6	1	1.33e-3
clip-2nail	3	8	393	219	194	35962				1.06e-2
L shape, 3D	6	1	0	1	1	1087	200^3x90^3	5.83e12	1	1.87e-10
T in jail	6	6	5	4	2	9192	$100x75^2$ x251^2x157	5.56e12	1	1.64e-9
Π in jail	6	89	85	30	23	55586	100^2x75 x251^2x157	7.42e12	1	7.51e-9

TABLE III
PERFORMANCE STATISTICS OF SANDROS PLANNER FOR MULTIPLE RIGID OBJECTS

example	dof	time	N_{node}	N_{local}	N_{seq}	N_{dist}	grid size	N_{grid}	λ	$\mathcal{E}$
2 people	6	6	13	6	3	7388	$(70^2$x$120)^2$	3.46e11	1	2.14e-8
3-spike assembly	9	6	3	2	1	7081	$(70$x54x$120)^3$	9.33e16	1	7.59e-14
2-part insertion	4	4	105	48	42	5541	$(120$x$90)^2$	1.17e8	1	4.62e-5
	6	82	1135	925	899	101605	$(120$x90x$120)^2$	1.68e12	1	6.05e-8
6 squares	18	207	45	25	16	226771	$(70^2$x$120)^6$	4.13e34	1	5.48e-30
2 furniture	12	113	1695	477	339	67061	$(70^2$x25x$80^3)^2$	3.93e21	1	1.70e-17
3 furniture	18	26908	65735	25134	20912	7378735	$(70^2$x25x$80^3)^3$	2.47e32	1	2.99e-26

in [31] will take a long time to find a feasible configuration in the tunnel via random sampling.

If the C-space has a big dead-end space, i.e., a trap, and the robot has to backtrack, Kondo's algorithm is somewhat inefficient due to the lack of an *explicit* backtracking mechanism. Our planner and the planners in [23] and [31] provide backtracking mechanisms with subgoals, visibility graph or raodmap, respectively, and can better handle traps. Our planner and that in [23], however, do not generalize well for a manipulator with a tree structure, but the planners in [4], [31], and [34] have no problem. For a general application, the best way is to run all of these in parallel and take the first satisfactory solution.

A. Manipulator

Fig. 3 shows three problems of increasing difficulty with a 2-link planar robot. Solution motions and the subgoal graphs in the C-space are shown along with the C-space obstacles [Fig. 3(d)]. The size of the subgoal graph increases with the problem difficulty. The real power of SANDROS shows

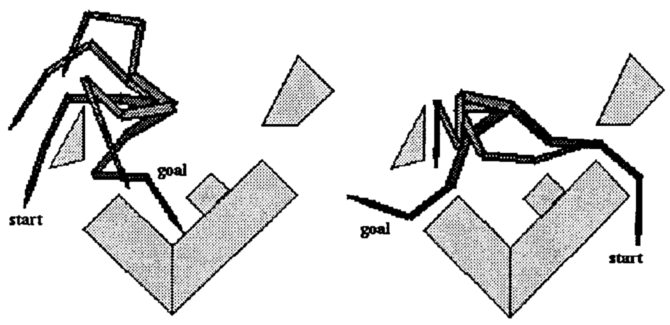


Fig. 4. Two 4-DoF problems solved in 8 s (left) and 14 s (right).

in solving problems with higher DoFs. Fig. 4 shows 4-DoF problems, which are solved in 8 and 14 s. Planning motion of a 5-DoF AdeptOne robot to move an L shape out of a wicket to the left side of the table takes 22 s (Fig. 5). Pulling a stick from behind two vertical posts with a 6-DoF Puma robot is shown in Fig. 6 (5 s). Finally, the 9-DoF problem shown in Fig. 7(c) takes about 15 min, showing the gradual

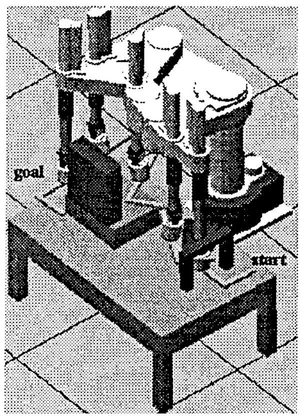


Fig. 5. A 5-DoF Adept robot is moving an L-shaped object out of a wicket and places it on the left side of the table. The computation time is 22 s.

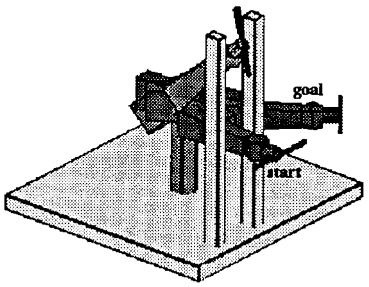


Fig. 6. The SANDROS planner took 5 s to find the motion to move a stick through the two vertical posts.

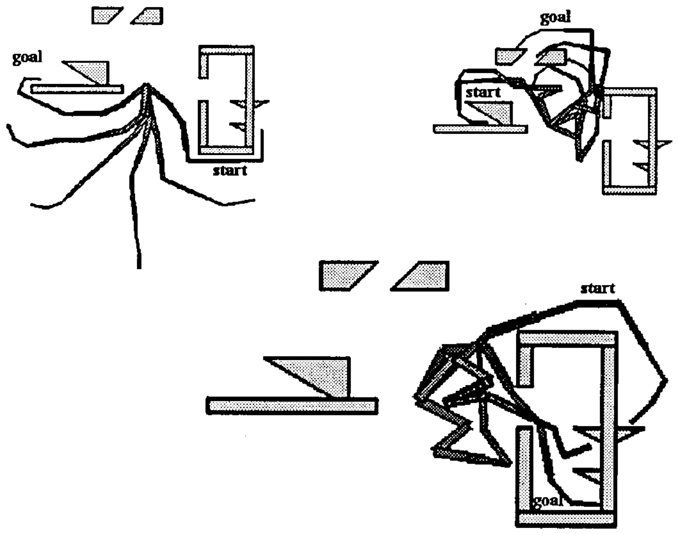


Fig. 7. Easy, intermediate, and hard problems involving a nine-link planar robot. It took SANDROS 10 s (top left), 10 min (top right) and 16 min (bottom) to solve these problems, respectively.

increase of computation time with DoF. In the examples above where obstacles and robots have about ten to 40 faces, the time for one distance computation was roughly 1 ms. For typical 5 and 6-DoF problems of moderate difficulty (Figs. 5 and 6), SANDROS shows near real-time performance. If the geometric complexities of robots and environments increase, or for very difficult problems, SANDROS will need a 10 to 100 times faster computer for near real-time performance.

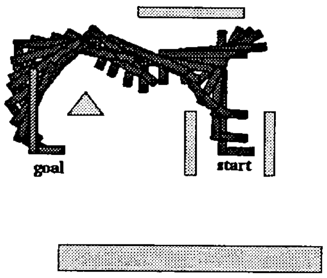


Fig. 8. Moving an L shape. SANDROS took 1 s to solve this problem.

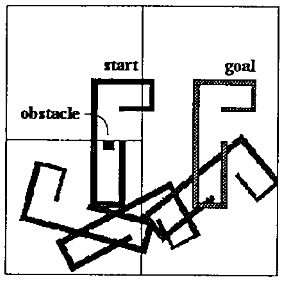


Fig. 9. Unhooking a paper clip from a nail (small square). The computation time is 1 s. The rectangular partition shows subgoals generated by SANDROS.

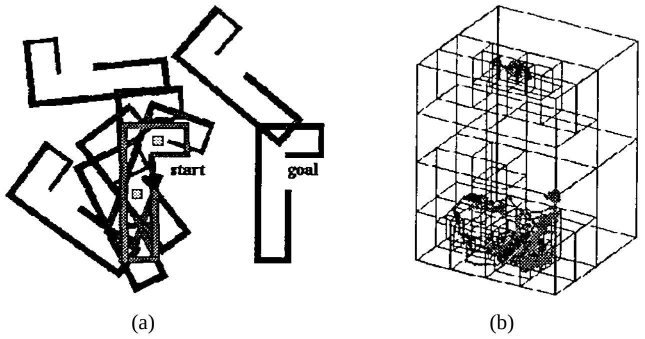


Fig. 10. Moving a paper clip away from two nails (small squares) took 8 s for SANDROS. The cell subgoals and the traces of the local planner are shown in (b).

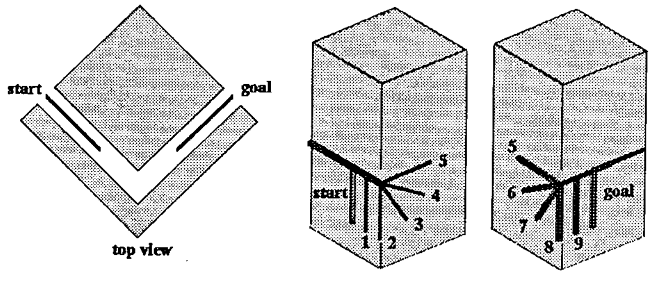


Fig. 11. Moving an L shape around the corner of a hallway (front walls not shown). It took one call to the local planner to solve this problem.

B. Rigid Object

Fig. 8 shows a simple 2-D problem, solved in 1 s. Fig. 9 shows the problem of unhooking a half-open paper clip from a nail (small square). The partition of the space into rectangles shows the cell-subgoals used by SANDROS. When we add another nail, the problem becomes considerably harder, and Fig. 10(b) shows a more complex partition of C-space computed by SANDROS. Fig. 11 shows how to move an L

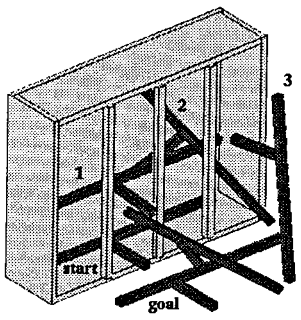


Fig. 12. Moving a T shape out of a jail. The computation time is 6 s.

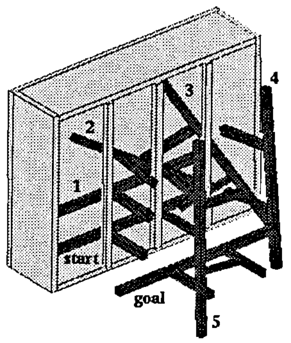


Fig. 13. Moving a Pi shape out a jail. The computation time is 89 s.

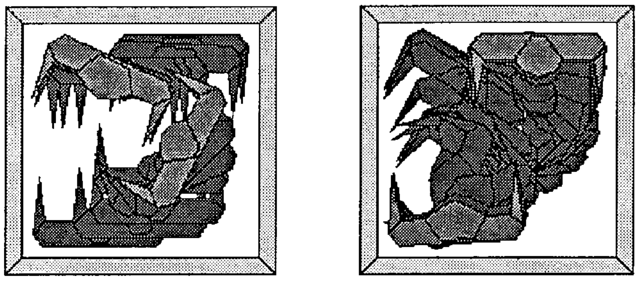


Fig. 14. Two people exchanging their positions in a small room. This 6-DoF problem was solved in 6 s.

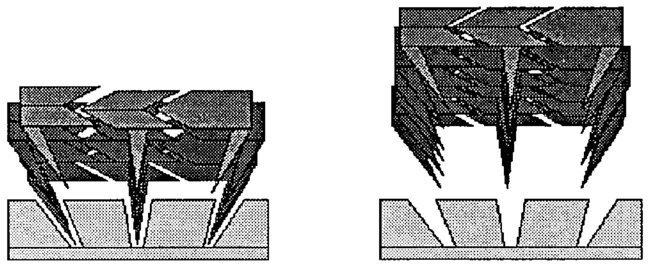
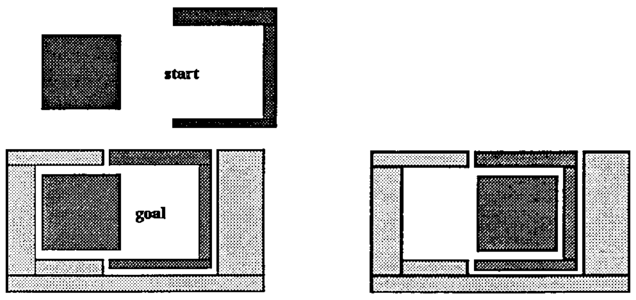
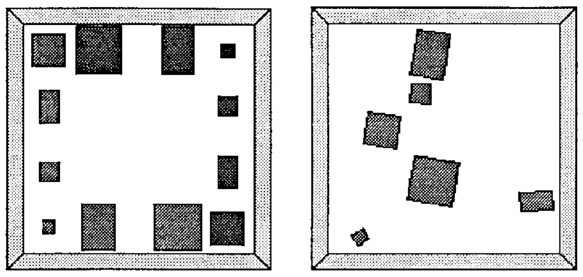


Fig. 15. An assembly requiring simultaneous movements of three objects. This 9-DoF problem was solved in 6 s.



(a) (b)

Fig. 16. This nonmonotone assembly needs the crucial configuration in (b) to be assembled. It is solved in 4 s with only translation (4 DoF), and in 82 s when rotation is also allowed (6 DoF).



(a) (b)

Fig. 17. (a) Moving six mobile robots to the opposite side of the room. The computation time is 207 s. (b) Snapshot at a point along the solution path.

abound the corner of a hallway (front walls not shown). It turns out that SANDROS can solve this problem with just one call to the local planner. Figs. 12 and 13 show how to move a T and a Pi shapes out of a jail cell (6 and 89 s). Note that real-world problems do not get much harder than these problems. Otherwise, the robot should probably remove some obstacles.

C. Multiple Rigid Objects

For multiple movers' problem, SANDROS is most useful when the total DoF of robots is not too large (< 13) and the solution requires simultaneous motions of robots. If robots can be moved one at a time, it is better to decompose the problem into a sequence of single mover's problems. Fig. 14 shows the top view of two people exchanging their positions in a small room (6 s). The spike example in Fig. 15 is taken from [43] which was used to show that there is an assembly requiring an arbitrarily many hands to assemble. (We added a finite tolerance between the baseboard and the spikes since our planner considers contacts as collisions.) A nonmonotone assembly problem of inserting two parts in Fig. 16(a) is solved in 4 s when only translational motion is allowed (4 DoF). If the rotation is included (6 DoF problem), it is solved in 82 s, most of time being spent on finding the crucial configuration in Fig. 16(b).

Fig. 17 shows six mobile robots trying to go to the opposite side of a room (an 18-DoF problem, 207 s). Fig. 17(b) shows a snap shot of the room at one point of the solution motion. The motions are erratic since the planner tries to move them away from one another. The problems of rearranging two and three pieces of furniture in 3-D are shown in Figs. 18 (2 min) and 19 (7.5 h).

V. Conclusion

We have presented a dynamic graph search algorithm called SANDROS, and applied it to the gross motion planning of a manipulator, a rigid object, and multiple rigid objects. It is a resolution-complete algorithm with computation time

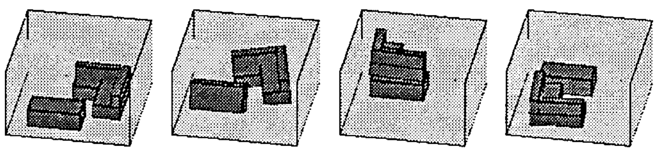


Fig. 18. Rearranging two pieces of furniture (the ceiling and the front wall are not shown). It took SANDROS 2 min to solve this 12-DoF problem.

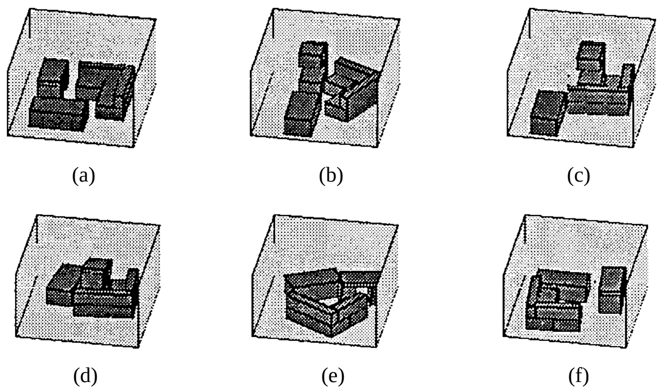


Fig. 19. Rearranging three pieces of furniture (the ceiling and the front wall are not shown). It took SANDROS 7.5 h to solve this 18-DoF problem.

commensurate with problem difficulty, and is one of the most efficient and complete motion planners developed to date. SANDROS algorithm is a judicious combination of several important ideas developed in motion planning over the past decade. They are multiresolution search using hierarchical cell division of the C-space [7], a local planner based on a potential field [32], [27], a two-level search scheme in [18], sequential computation of collision-free joint-angle ranges for a manipulator [40], an efficient distance computation [19], and a bidirectional search [27]. To obtain a planner significantly faster than ours would seem to require using massively parallel machines [10], or taking a totally different approach utilizing sensors or additional knowledge.

SANDROS planner has many applications. For manipulators, it can be used *on-line* in unstructured environments such as in exploration or waste-site cleanup. For structured environments such as in manufacturing, it can automatically program robots *off-line* in place of manual programming. Our planner for a rigid object can be used for navigating mobile robots, and to plan motions to assemble mechanical products. Our planner for multiple rigid objects is especially useful for cases where simultaneous motions of robots are required. Our planners show the best performance when there are less than 10 DoF. For low-DoF problems, brute-force algorithms also perform well, while for problems with many DoF (e.g., >10), the algorithms in [4] and [42] are more suitable.

There are many future directions in motion planning research. For the model-based approach, planning motions with contacts will benefit assembly planning research. Polyhedral convex cones [21] are the most popular tool for this. For the sensor-based approach, selectively building models of the environment at the right resolution will be a good research direction. Purely sensor-based algorithms will be suitable for reflex motions, while some model of the world will be necessary for global planning [41]. Developing a faster distance routine can reduce motion planning time even further. For example, the thresholded distance [52] does not compute the exact distance if it determines the distance is greater than a threshold, and thereby reducing the computation time. Since the collision avoidance need not be done unless obstacles are closer than some threshold (D_{safe} in Section IV, fourth paragraph), the thresholded distance is particularly useful for motion planning.

Constraints such as nonholonomic constraints can be incorporated into motion planning by constraining robot motion in the local planner [48]. To include dynamics, the search has to be done in the position-velocity space, or the phase space [15]. The knowledge-based approach for motion planning is explored to some extent. For example, spatial reasoning is incorporated into motion planning in [30]. Issues like knowledge representation and plan generation must be resolved before a knowledge-based motion planner can be developed.

Acknowledgment

The authors would like to thank E. G. Gilbert, University of Michigan, for providing the software for computing distance between polyhedra, and the reviewers of this paper, including D. R. Strip, R. C. Brost, and P. G. Xavier.

References

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] F. Avnaim, J. D. Boissonnat, and B. Faverjon, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles," in *Proc. IEEE Int. Conf. Robot. Automat.*, Philadelphia, PA, 1988, pp. 1656–1661.

[3] J. Barraquand and P. Ferbach, "Path planning through variational dynamic programming," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Diego, CA, 1994, pp. 1839–1846.

[4] J. Barraquand and J. C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom," in *Proc. IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, 1990, pp. 1712–1717.

[5] ——, "Robot motion planning: A distributed representation approach," *Int. J. Robot. Res.*, vol. 10, pp. 628–649, 1991.

[6] P. Bessiere, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer, "The "Ariadne's Clew" algorithm: Global planning with local method," in *Proc. Int. Conf. Intell. Robot. Syst.*, 1993, pp. 1373–1380.

[7] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for findpath with rotation," *Int. Joint Conf. Artif. Intell.*, Karlsruhe, Germany, 1983.

[8] S. J. Buckley, "Fast motion planning for multiple moving robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Scottsdale, AZ, 1989, pp. 322–326.

[9] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[10] D. Challou, M. Gini, V. Kumar, and C. Olson, "Very fast motion planning for dexterous robots," in *Proc. IEEE Int. Symp. Assembly Task Plan*, Pittsburgh, PA, 1995, pp. 201–206.

[11] P. C. Chen, "Improving path planning with learning," in *Proc. 9th Int. Conf. Mach. Learn*, 1992, pp. 55–61.

[12] P. C. Chen and Y. K. Hwang, "SANDROS: A motion planner with performance proportional to task difficulty," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nice, France, 1992, pp. 2346–2353.

[13] Y. K. Hwang and P. C. Chen, "A heuristic and complete planner for the classical mover's problem," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nagoya, Japan, May 1995, pp. 729–736.

[14] B. Donald, "Motion planning with six degrees of freedom," tech. rep. AI-TR-791, Arti. Intell. Lab., Mass. Inst. Technol., Cambridge, MA, 1984.

[15] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. ACM*, vol. 40, no. 5, pp. 1048–1066, Nov. 1993.

[16] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Francisco, CA, 1986, pp. 1419–1424.
[17] B. Faverjon, "Obstacle avoidance using an octree in the configuration space," in *Proc. IEEE Int. Conf. Robot. Automat.*, Atlanta, GA, Mar. 1984, pp. 1152–1159.
[18] B. Faverjon and P. Tournassoud, "A local approach for path planning of manipulators with a high number of degrees of freedom," in *Proc. IEEE Int. Conf. Robot. Automat.*, Raleigh, NC, Mar. 1987, pp. 1152–1159.
[19] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE J. Robot. Automat.*, vol. 4, pp. 193–203, Apr. 1988.
[20] A. J. Goldman and A. W. Tucker, "Solving findpath by combination of goal-directed and randomized search," in *Proc. IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, May 1990, pp. 1718–1723.
[21] ——, "Polyhedral convex cones," in *Linear Inequalities and Related Systems, Annals of Mathematics Studies 38*, H. W. Kuhn and A. W. Tucker, Eds. Princeton, NJ: Princeton Univ. Press, pp. 19–39.
[22] K. K. Gupta, "Fast collision avoidance for manipulator arms: A sequential search strategy," in *Proc. IEEE Int. Conf. Robot. Automat.*, Cincinnati, OH, May 1990, pp. 1724–1729.
[23] ——, "A 7-DoF practical motion planner based on sequential framework: Theory and experiments," in *Proc. IEEE Int. Symp. Assem. Task Plan.*, Pittsburgh, PA, 1995, pp. 213–218.
[24] J. Hopcroft, D. Joseph, and S. Whiteside, "Movement problems for 2-dimensional linkages," *SIAM J. Comput.*, vol. 13, no. 3, pp. 610–629, 1984.
[25] J. Hopcroft and G. T. Wilfong, "Reducing multiple object motion planning to graph searching," *SIAM J. Comput.*, vol. 15, no. 3, pp. 768–785, 1986.
[26] Y. K. Hwang, "Motion planning for multiple moving objects," in *Proc. IEEE Int. Symp. Assem. Task Plan*, Pittsburgh, PA, 1995, pp. 400–405.
[27] Y. K. Hwang and N. Ahuja, "Potential field approach to path planning," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 23–32, Feb. 1992.
[28] ——, "Gross motion planning—A survey," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 219–292, Sept. 1992.
[29] Y. K. Hwang and P. C. Chen, "A heuristic and complete planner for the classical mover's problem," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nagoya, Japan, May 1995, pp. 729–736.
[30] Y. K. Hwang, P. C. Chen, and P. A. Watterberg, "Interactive task planning through natural language," in *Proc. IEEE Int. Conf. Robot. Automat.*, Minneapolis, MN, May 1995, pp. 24–29.
[31] L. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmap for path planning in high dimensional configuration space," *IEEE Trans. Robot. Automat.*, vol. 12, pp. 566–580, Aug. 1996.
[32] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, St. Louis, MO, 1985, pp. 500–505.
[33] Y. L. Koga and J. C. Latombe, "On multi-arm manipulation planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Diego, CA, 1994, pp. 945–952.
[34] K. Kondo, "Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration," *IEEE Trans. Robot. Automat.*, vol. 7, pp. 267–277, June 1991.
[35] J. C. Latombe, *Robot Motion Planning*. New York: Kluwer, 1991.
[36] S. M. LaValle and S. A. Hutchinson, "Path selection and coordination for multiple robots via Nash equilibria," in *Proc. IEEE Int. Conf. Robot. Automat.*, San Diego, CA, 1994, pp. 1847–1852.
[37] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," *Comput. Graph.*, vol. 24, no. 4, pp. 327–335, Aug. 1990.
[38] Y. H. Liu, S. Kuroda, T. Naniwa, H. Noborio, and S. Arimoto, "A practical algorithm for planning collision-free coordinated motion of multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Scottsdale, AZ, 1989, pp. 1427–1432.
[39] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, 1979.
[40] T. Lozano-Pérez, "A simple motion-planning algorithm for general robot manipulators," *IEEE J. Robot. Automat.*, vol. RA-3, pp. 224–238, June 1987.
[41] V. J. Lumelsky, "Incorporating body dynamics into the sensor-based motion planning paradigm, the maximum turn strategy," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nagoya, Japan, 1995, pp. 1637–1642.
[42] A. A. Maciejewski and C. A. Klein, "Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments," *Int. J. Robot. Res.*, vol. 4, no. 3, pp. 109–117, 1985.
[43] B. K. Natarajan, "On planning assemblies," in *Proc. 4th ACM Symp. Computat. Geometry*, 1988, pp. 299–308.
[44] C. O'Dúnlaing and C. K. Yap, "A retraction method for planning the motion of of a disc," *J. Algorithms*, vol. 6, pp. 104–111, 1982.
[45] B. Paden, A. Mees, and M. Fisher, "Path planning using a Jacobian-based freespace generation algorithm," in *Proc. IEEE Int. Conf. Robot. Automat.*, Scottsdale, AZ, 1989, pp. 1732–1737.
[46] C. Qin, S. Cameron, and A. McLean, "Toward efficient motion planning for manipulators with complex geometry," in *Proc. IEEE Int. Symp. Assem. Task Plan*, Pittsburgh, PA, 1995, pp. 207–212.
[47] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Proc. 20th IEEE Symp. Foundations Comput. Sci.*, 1979, pp. 421–427.
[48] P. Švestka and M. H. Overmars, "Coordinated motion planning for multiple car-like robots using probabilistic road maps," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nagoya, Japan, May 1995, pp. 1631–1636.
[49] J. T. Schwartz and M. Sharir, "On the piano movers' problem: II. Techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*. New York: Academic, 1983, pp. 298–351.
[50] R. H. Wilson and T. Matsui, "Partitioning an assembly for infinitesimal motions in translation and rotation," in *Proc. Int. Conf. Intell. Robots Syst.*, 1992, pp. 1311–1318.
[51] C. K. Yap, "How to move a chair through a door," *IEEE J. Robot. Automat.*, vol. RA-3, pp. 172–181, 1987.
[52] P. A. Watterberg, P. G. Xavier, and Y. K. Hwang, "Path planning for everyday robotics with SANDROS," in *Proc. IEEE Int. Conf. Robot. Automat.*, Albuquerque, NM, 1997, pp. 1171–1176.
[53] D. Y. Yeung and G. A. Bekey, "A decentralized approach to the motion planning problem for multiple mobile robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, Raleigh, NC, 1987, pp. 1779–1784.

Pang C. Chen received the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1989.

He presently works at the Sandia National Laboratories, Albuquerque, NM. His research interests include combinatorial algorithms, machine learning, and massively parallel computing. His personal lifelong challenge is to develop a Go-playing program that can defeat him using AI techniques and distributed computing resources.

Dr. Chen is a co-recipient of the 1994 Research & Development 100 Award for his work on motion planning.

Yong K. Hwang (M'87) received the Ph.D. degree in electrical engineering from the University of Illinois, Urbana-Champaign, in 1988.

Since then, he has been working at the Sandia National Laboratories, Albuquerque, NM, and is currently a Principal Member of Technical Staff. His research interests include robotics, vision, artificial intelligence, and man-machine interface.

Dr. Hwang is a co-recipient of the 1994 Research & Development 100 Award for his work on motion planning.