Preprint: Special Issue on „Motion Planning“ of the Journal of Robotic Systems, 2001 © 2001 Wiley & Sons, Inc.

Point-to-Point and Multi-Goal Path Planning for Industrial Robots

Christian WURLL
AMATEC Robotics GmbH
Landsbergerstraße 63a
D-82110 Germering, Germany
e-mail: Christian.Wurll@amatec.de

Dominik HENRICH*
Embedded Systems and Robotics Laboratory (RESY), Faculty of Informatics, Building 48,
University of Kaiserlautern,
D-67653 Kaiserslautern, Germany
e-mail: henrich@informatik.uni-kl.de

This article presents contributions in the field of path planning for industrial robots with 6 degrees of freedom. This work presents the results of our research in the last 4 years at the Institute for Process Control and Robotics at the University of Karlsruhe. The path planning approach we present works in an implicit and discretized C-space. Collisions are detected in the Cartesian workspace by a hierarchical distance computation. The method is based on the A search algorithm and needs no essential off-line computation. A new optimal discretization method leads to smaller search spaces, thus speeding up the planning. For a further acceleration, the search was parallelized. With a static load distribution good speedups can be achieved. By extending the algorithm to a bidirectional search, the planner is able to automatically select the easier search direction. The new dynamic switching of start and goal leads finally to the multi-goal path planning, which is able to compute a collision-free path between a set of goal poses (e.g., spot welding points) while minimizing the total path length.*

Keywords: industrial robots, path planning, on-line algorithms, search algorithms, graph search, bidirectional search, discretization, distributed and parallel processing

1. Introduction

Today’s production lines usually consist of multiple robots, interacting with a wide range of equipment and fixtures. Programming these capital intensive installations can be done off-line in powerful robot simulation systems. Off-line programming is still a complex task and the resulting programs strongly depend on the programmer’s capabilities.[1] Let us, for instance, consider a spot welding task in which a robot has to reach several spot welding points. In this scenario, the main goal of the programmer is to generate a collision-free robot path, which can be executed as quickly as possible to achieve short cycle times, thus increasing the total throughput. Depending on the problem complexity, even an experienced programmer needs a significant amount of time to find a solution, and the solution most likely is suboptimal.
It is difficult to choose an optimal sequence and, at the same time, find a collision-free path between two spot welding points. Yet, neither in the current state of the art nor in the existing

* To whom correspondence should be addressed.
This work was performed at the Institute for Process Control and Robotics (IPR), Computer Science Department, University of Karlsruhe (TH), P.O. Box 6980, D-76128 Karlsruhe, Germany

robot simulation tools, like ROBCAD, IGRIP or CATIA, are tools available to solve the multi-goal path planning problem.

The main part of the *multi-goal path planning* is finding a collision-free path. The issue of robot path planning has been studied for several decades and many important contributions have been made to the problem.[2] Point-to-point (PTP) path planning algorithms, which can find a collision-free path from a start configuration (point) to a goal configuration (point) are of great theoretical interest, but are rarely used in practice because of their computational complexity.[4] In the last 2 years a few new PTP path planning approaches have been published, which promise good results (see, e.g., Baginski[5] and Chen and Hwang[6]).

The multi-goal (MTP) path planning problem, which computes a collision-free path as well as the optimal sequence, has not yet been considered. In our opinion, solving the MTP problem can improve the programs generated off-line and, therefore, reduce the total programming time.

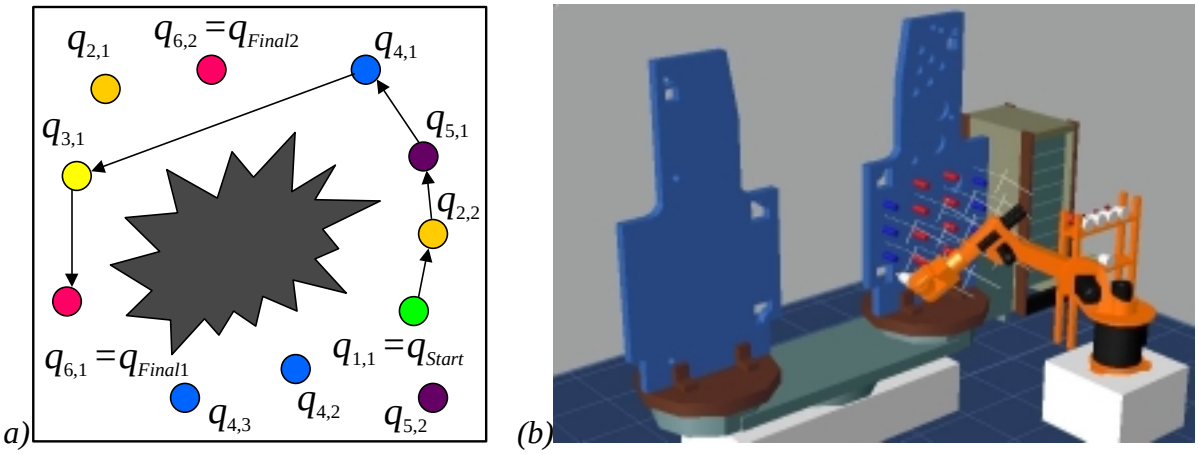


Figure 1. (a) Illustration of a MTP problem in a two-dimensional C-space with one static obstacle (star) and several different goal configurations (dots). Dots with the same first index belong to the same pose, the arrows indicate the optimal sequence, (b) Screenshot of the MTP problem GRINDING in ROBCAD with 21 different goal poses (grinding points).

In this article, we present contributions to the field of PTP and MTP path planning, which are the results of our research in the last 4 years at the Institute for Process Control and Robotics at the University of Karlsruhe. More precisely, we assume we are given an industrial robot, usually with 6 degrees of freedom (DOF), and a set of static obstacles. Both, the robot and the obstacles are provided as computer-assisted design (CAD) models. Additionally, a set Q of different goal poses representing the position and orientation of the robot's tool center point in the work space (W-space) is given. Because of ambiguous inverse kinematics of the robot, a pose can be reached by several different configurations in the configuration space (C-space). The MTP problem is stated as follows: compute a collision-free path between these poses and find the optimal pose sequence, thus reaching every pose at least once while minimizing the total path length. If Q has only two elements, then the problem represents a standard PTP path planning problem (Figure 1).

The remainder of the article discusses the following questions: which point-to-point path planning method will work reasonably fast in industrial applications (Section 2)? How can this basic approach be accelerated (Section 3)? What advantages does a multi-goal path planning method have in contrast to a point-to-point path planning method (Section 4)? And finally, what conclusions can be drawn (Section 5)?

2. Point-to-Point path planning

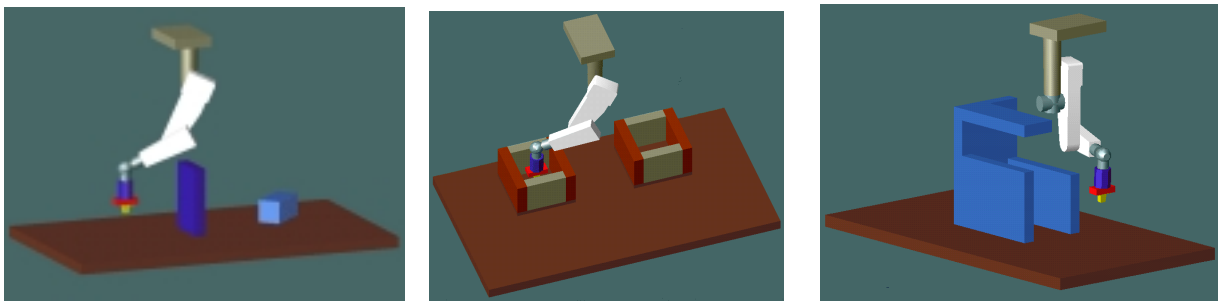
Most off-line path planners are based on some explicit representation of the *free C-space*. This representation can either be retrieved by transforming the obstacle into the C-space and

approximating the free space or by randomly sampling the C-space and interconnecting the samples by collision-free links. Both approaches are time consuming and not suited for on-line calculations, especially if a full geometric CAD model for the robot and the obstacles is used. To avoid these time consuming calculations, one can search in an implicitly represented C-space and detect collisions in the workspace. For searching in the implicit C-space, any best-first search mechanism can be applied. We choose a variation of the well known A* search algorithm.[3] The C-space is discretized and all the robot configurations are represented by nodes building up the search space.

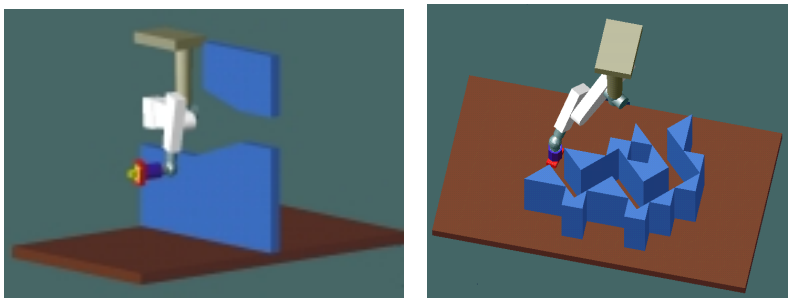
2.1. Benchmark problems

As a basis for an objective evaluation of the path planner, a set of test environments with corresponding problem specification (*benchmark problem*) is used. Because the planner might use different robots and the robots might differ in their construction (e.g., geometry and kinematics), one cannot compare a problem specification for a robot *A* in a test environment with the same problem specification for a robot *B* in the same test environment. Therefore, the test environments are not specified in the workspace but schematically in a two-dimensional (2D) C-space with increasing level of difficulty.

The levels of difficulty called SIMPLE, STAR, TRAP, and BOTTLENECK were presented in Hwang.[7] A new level of difficulty, called DETOUR, is introduced, which includes a shorter path near to obstacles and a longer path away from obstacles. This enables us to investigate the path planner's ability to find a reasonable tradeoff between finding a long path that can be executed fast, and a short path that requires moving at a lower speed.



(a) *(b)* *(c)*



(d) *(e)*

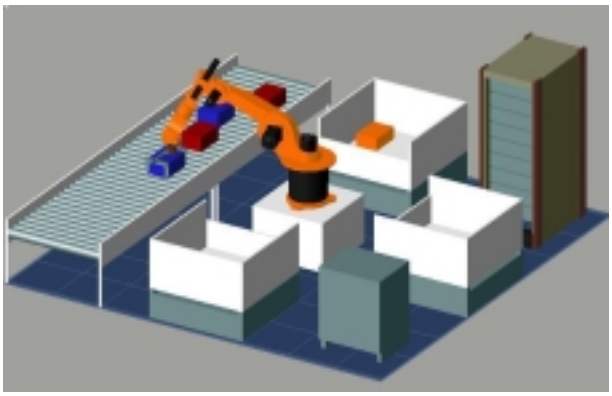
Figure 2. The 3-dimensional test environments for a Puma260 with 6 DOF: (a) SIMPLE, (b) STAR, (c) TRAP, (d) BOTTLENECK, and (e) DETOUR.

Based on these schemes, the corresponding test environments together with their problem specification have to be prepared for each type of robot.† Examples for the 6 DOF robot Puma260 in the robot simulation tool ROBCAD are shown in Figure 2.[8]

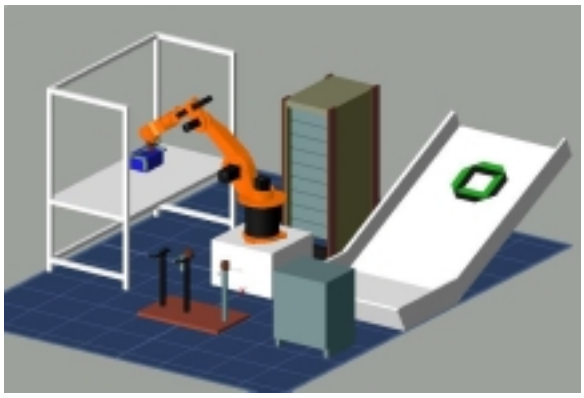
To evaluate the performance of the PTP approach for industrial conditions, three additional applications (SORT, TRANSFER, and PRESS) have been investigated (Fig. 3). In all three problems, the work cell contains more obstacles and more complex robot models

† The data for these benchmark problems can be downloaded from the Web page at http://wwwipr.ira.uka.de/~paro/skalp/.

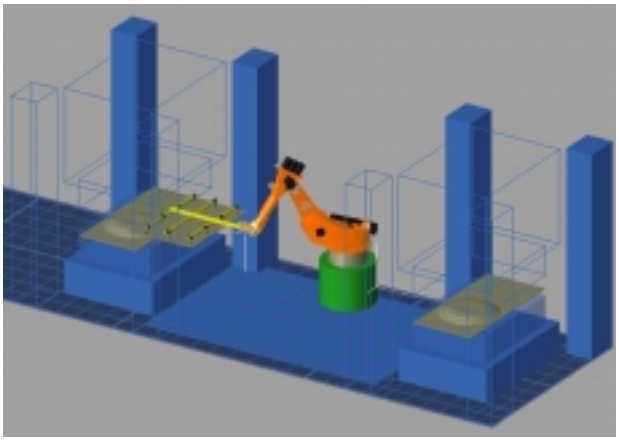
(Kuka KR 15 and Kuka KR 100P), thus, increasing the computational demand for the collision detection.



(a)



(b)



(c)

Figure 3. Screenshot of three industrial applications: SORT (a), TRANSFER (b), and PRESS (c).

2. 2. C-Space Discretization

As the path is planned in the discretized C-space, deciding the level of resolution for the discretization is an important issue. A too fine discretization will increase the search space; a too coarse discretization may result in failing to find a path even if one exists. Formally, for the *i*th coordinate q_i of the C-space, let N_i be the number of intervals along q_i. Then we can determine N_i by‡

$$N_i = \left\lfloor \frac{q_i^{\max} - q_i^{\min}}{\Delta q_i} \right\rfloor,$$

where $q_i^{\max}$ and $q_i^{\min}$ are the limits of joint motions and Δq_i is the resolution of joint *i*. The complete search space size, *N*, is then the product of all N_i. The question is now how to determine the Δq_i.

We have investigated three different methods to determine the discretization resolution.[9] In the simplest method, the user specifies a *uniform discretization* for all joints, thus $\Delta q = c$ for some constant *c*. With a reasonable joint resolution of 1°, the uniform discretization results in huge C-spaces. To avoid the huge search space produced by the uniform discretization, usually a *heuristic discretization* is applied. Here, reasonable Δq_i are estimated by the user to balance the resulting Cartesian movement Δx_i in W-space when different joints *i* are moved for Δq_i in C-space (Fig. 4a).

‡ Here, $\lfloor x \rfloor$ denotes the next smaller integer of *x*.

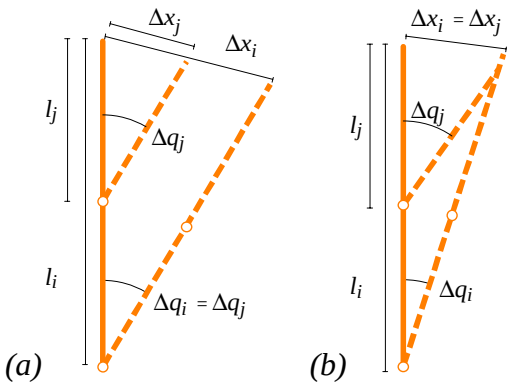


Figure 4. (a) The uniform discretization ($\Delta q_i = \Delta q_j$) results in different Cartesian movements $\Delta x_i \neq \Delta x_j$ when different joints i, and j are moved. (b) The optimal discretization results in equal maximum Cartesian movement $\Delta x_i = \Delta x_j$ when different joints are moved

Instead of having a uniform or a heuristic resolution, an *optimal discretization* can be calculated. Therefore, the resolution along each coordinate is set according to the maximum movement of the robot's end-effector (Fig. 4b). Analytically, this can be achieved by setting

$$\Delta q_i = 2 \arcsin\left(\frac{\Delta x_{\max}}{2 l_i}\right),$$

where l_i is the distance between the center of joint *i* to the farthest point the end-effector can reach, and $\Delta x_{\max}$ is a preset user defined distance the robot may move in one step along the coordinate.[10] Altogether, the optimal discretization results in a Cartesian movement Δx_i of joint *i* which meets the condition $\Delta x_i \leq \Delta x_{\max}$.

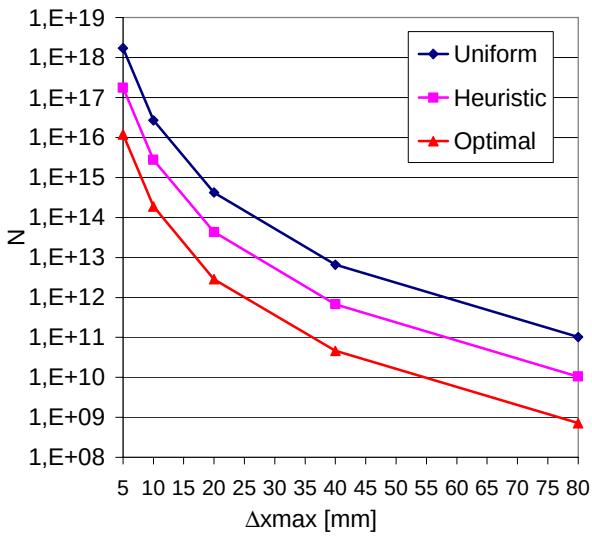


Figure 5. Size N of discrete C-space versus maximal Cartesian movement $\Delta x_{\max}$ for the different discretization methods.

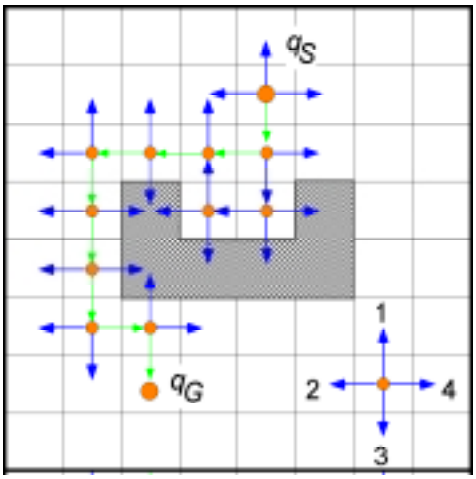


Figure 6. A 2D illustration of the A search in the implicit C-space from the start configuration q_S to the goal configuration q_G. The dots indicate the configurations examined (stored in CLOSED) and the arrows point to the corresponding successors (stored in OPEN).*

The resulting search space sizes depending on the maximal Cartesian movement $\Delta x_{\max}$ are shown in Figure 5. Compared with the homogeneous discretization, the search space with the optimal discretization is about 100 times smaller. Additionally, the minimum W-space resolution can be improved, without changing the size of the search space (C-space).

2.3. Sequential Search

The search algorithm maintains a CLOSED list of the nodes that have been expanded and an OPEN list of the nodes that have been generated but not yet expanded. The algorithm begins with the start node q_S in the OPEN list. At each iteration, a node in the OPEN list with the

minimum heuristic evaluation is expanded, generating all of its successors and is placed on the CLOSED list. An evaluation function *f(n)* is applied to all collision-free successors *n*, and they are placed on the OPEN list sorted by their heuristic values *f*. The search continues until a goal node q_G is chosen for expansion or the OPEN list is empty. In the latter case, the algorithm stops with no solution. Contrasting to the original A*, here no reopening of nodes in CLOSED is performed. Also, colliding successors may be inserted in OPEN. But both modifications lead to an enormous acceleration of the search (Figure 6).
An evaluation function $f(n) = (1–w)\, g(n) + w h(n)$ is used, where $g(n)$ is the number of nodes of the path from the start node q_S to node *n*, and $h(n)$ is the distance in C-space from node *n* to the goal node q_G. Increasing the weight $w \in [0, 1]$ beyond 0.5 generally decreases the number of investigated nodes while increasing the cost of the solutions generated. To improve the on-line capabilities of the path planner, our search is strongly directed to the goal by setting $w = 0.99$.[11] Of cause, this is equivalent to ignoring the measure $g(n)$ of accumulated path distance, which in its turn amounts to leaving out optimal paths in favor of efficiency. But in our experience, the paths found are still sufficiently short.

2.4. Collision Detection

Collisions are detected by a fast, hierarchical distance computation in the 3D workspace, based on the convex polyhedral model of the obstacles and the robot.[12] To avoid unnecessary calculations, the polyhedrons of the obstacles and the robot are divided into two classes (Figure 7).§ All polyhedrons in each class are additionally approximated by bounding-boxes and are hierarchically combined (Figure 8).

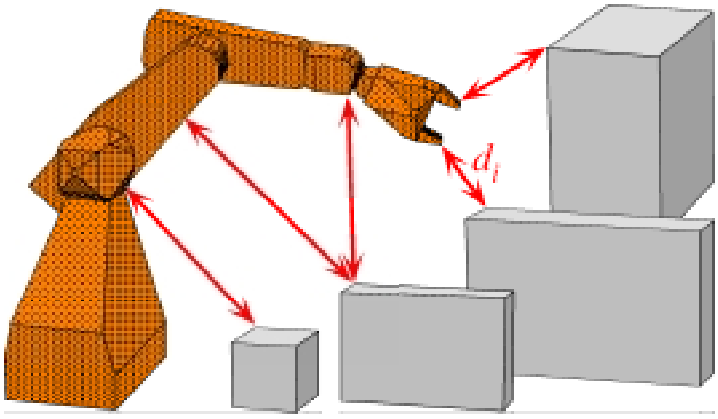


Figure 7. Collision detection in the workspace by computing the minimum distances d_i between the robot segment i and the obstacles.

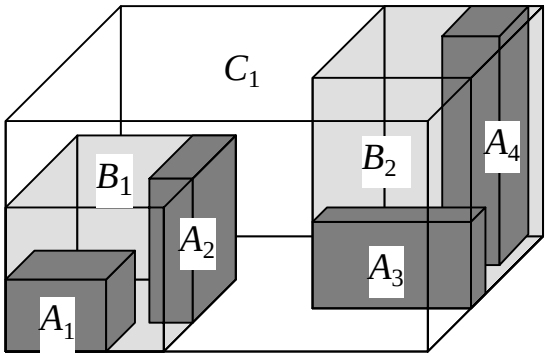


Figure 8. Four basic objects A_1 through A_4 and their hierarchical compositions B_1, B_2 and C_1.

During the collision detection between two collision classes the minimal distance needs to be computed. Therefore the two topmost hierarchy levels containing only one composition will be considered. At each level the minimal distance between all composition pairs is computed. If the distance between a pair is smaller than a desired threshold, then the members of that pair are substituted by their more precise representation at the level underneath. With these new pairs the computation is continued. The recursive algorithm terminates if either the distance among all viewed pairs is greater than the threshold or one pair at the lowest (most accurate) representation level of both classes is colliding.

Because the robot's configuration will change, the position and orientation of all objects in the robot's class and their compositions have to be computed at the beginning of every

§ To detect robot self-collisions between segments, the possible colliding segment pairs are determined off-line and treated on-line in the same way as the robot-obstacle polyhedron pairs.

distance calculation. To speed up this precalculation, the bounding-boxes of the robot's arm segments are computed based on the transformed *r*-cylinder approximations.** Only if a calculation at the lowest level is necessary, do the corresponding convex polyhedrons have to be transformed.[13]

As the Cartesian distance between the total robot and the obstacles cannot be efficiently used during the path planning, the robot-based distance calculation is extended to a segment-based one. In this case, for a robot with *n* DOF, the distance calculation results in *n* different minimal distances $d = [d_1, \ldots, d_n]^T$ between the *n* arm segments and the obstacles.[14]

To denote a configuration *q* in the C-space as "free", the half distance to the neighboring configurations must be free, which can formally be expressed by the *n*-dimensional vector $\varphi = 0.5 \cdot [\Delta q_1, \ldots, \Delta q_n]^T$. For small φ_i, this free space in joint space can be transformed into the maximum robot movement in W-space using the worst-case estimation††

$$\Delta d_i = s_i \cdot |\varphi_i| = \left(\sum_{j=i}^{n-1} l_j + r_{S,d} \right) \cdot |\varphi_i|$$

Here, the lengths l_i denote the distances between the joints θ_i and θ_{i+1}, and $r_{S,d}$ specifies the outer hull radius of the tool. To obtain a worst-case approximation of the maximum robot movement, the l_i are computed for a stretched arm configuration (Figure 9).

If, and only if, all calculated Cartesian distances d_i between the segments and the obstacles are larger than the maximum robot movement Δd_i, then the considered configuration is collision-free. To save further time-consuming distance calculations, every A*-node buffers the corresponding segment distances d_i. Before a new distance computation, the buffered distance vector is compared with the required robot movement necessary to reach the successor configuration. The new distance computation can then be omitted, if the segment distances are larger than the necessary maximum robot movement. With this segment-based distance buffering, the number of distance calculations can be reduced by about 65% on average, thus speeding up the path planning.

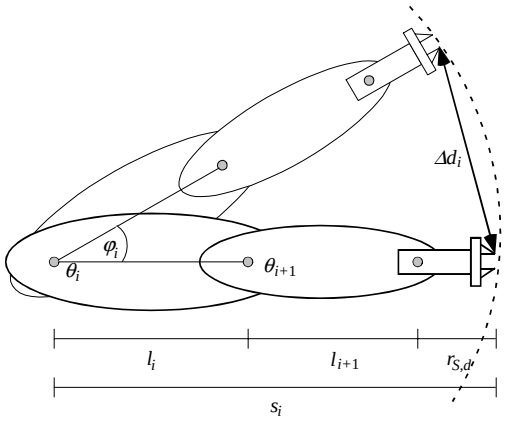


Figure 9. Length s_i of a stretched robot configuration for computing the maximum robot movement Δd_i for a rotation of joint θ_i about φ_i.

In addition to the fact that the distance calculation can terminate if a segment is colliding, a further acceleration is possible by calculating the distances of the single robot segments in a specific order. Experiments have shown that the initial segment order 4-6-3-5-2-1 for the benchmark problems described in Section 2 combined with an online reordering according to

** An *r*-cylinder consists of a cylinder with radius *r*, which is closed at its ends by two half-spheres. The two describing points can be transformed very quickly and *r*-cylinders are good approximations for arm segments.

†† Please note that $\sin(\varphi_i) = \varphi_i$ holds true for small $|\varphi_i|$.

the “last-hit-first” strategy leads to 1.5 to 3 times faster planning times for the Puma 260.

3. Extensions

To accelerate the planning, we have parallelized the basic search algorithm (Section 2). As the planning time may depend on the search direction (start to goal or goal to start), the extension to a bidirectional search enables the planner to automatically choose the easier direction. If the goal is specified in the W-space, it may be represented by several configurations in the C-space. By applying dynamic goal switching the search algorithm can automatically choose the easiest goal configuration.

3.1. Parallel Search

For parallelizing the A* algorithm, the configurations in OPEN and CLOSED must be accessible to all processors to distribute the work. Either these lists can be managed by one dedicated processor or each processor can maintain local lists. In a message passing system, each access to a global list would amount to an enormous communication effort. Thus the local method was preferred.[11]

The work distribution is the key aspect of parallelization. Therefore, for a robot with n DOF, the C-space is decomposed into n-dimensional hypercubes of size b in each dimension. For parallel processing, the hypercubes are cyclically mapped onto the p available processors by the following function: $M(\mathbf{q}) = 1 + \left(\sum_{i=1}^{n} \left\lfloor \frac{q_i}{\Delta q_i * b} \right\rfloor \right) \bmod p$.

According to the automatically computed discretization Δq_i, every configuration $\mathbf{q} = (q_1, \ldots, q_n)$ is mapped uniquely to one hypercube or to one processor. Thus the OPEN list of each processor contains configurations of the multiply mapped hypercubes (Figure 10).

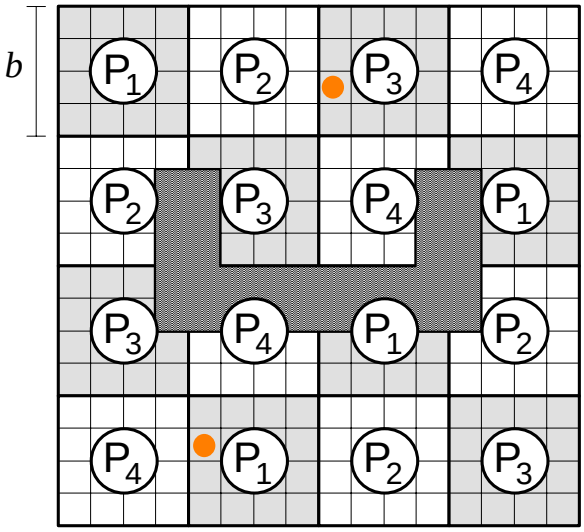


Figure 10. 2D illustration of the C-space decomposition in hypercubes of size b = 4 and the mapping of the hypercubes onto four processing units $P_1, \ldots, P_4$ mapping function M(q).

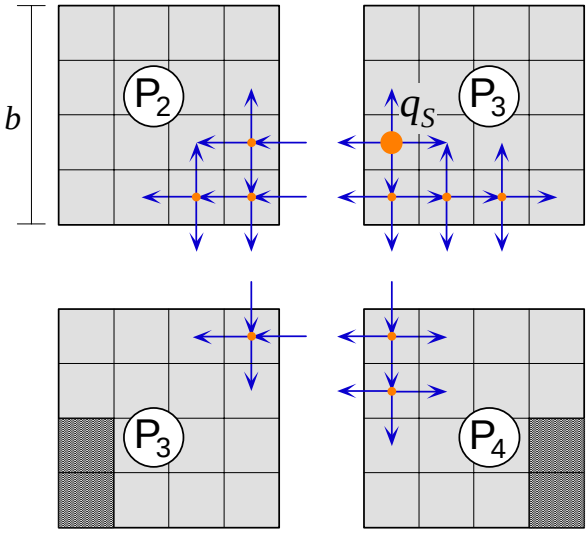


Figure 11. 2D illustration of the parallel search in four neighboring hypercubes taking the C-space decomposition of Figure 10 and according starting at q_S.

Each processor runs a local A* search beginning with the hypercube containing q_S. After the search has reached the hypercube boundaries, the expanded successors are sent to their corresponding processors. The configurations received are then inserted in a local OPEN list. As in the sequential version, at each iteration every processor expands the best configuration of OPEN until the list is empty or a goal node is chosen for expansion. In the former case, if

the OPEN lists of all processors are empty, the algorithm reports that there is no solution. In the latter case, the solution path is retraced across the hypercubes involved (Figure 11).

The performance of the parallel algorithm essentially depends on the load balancing mechanism, which can be influenced by modifying the cube size *b*. Considering the C-space decomposition, small sizes result in more cubes being mapped onto a single processor, thus implicating a good load distribution. In contrast, larger sizes worsen the load balance. On the other hand, smaller cubes leads to more messages, which may worsen the planning time according to the network capability. Thus, the specification of *b* will always be a tradeoff between a good load distribution and a minimum number of messages. Additionally, *b* influences behavior of search space exploration. For the benchmark problems considered, the best results are achieved with $6 \le b \le 16$ since few processors are idle and overall work load is small (Figure 12).

Based on a load distribution with $b = 16$, the parallel planning times for $P = [1,2,4,8]$ processors show how efficient the parallelization is. The planning times decrease with increasing numbers of processors (Pentium processor 133 MHz, 128 Mbytes).[15] With $P = 8$ processors, most planning times are under $T = 20$ s and the resulting speedup is linear (Figure 13).

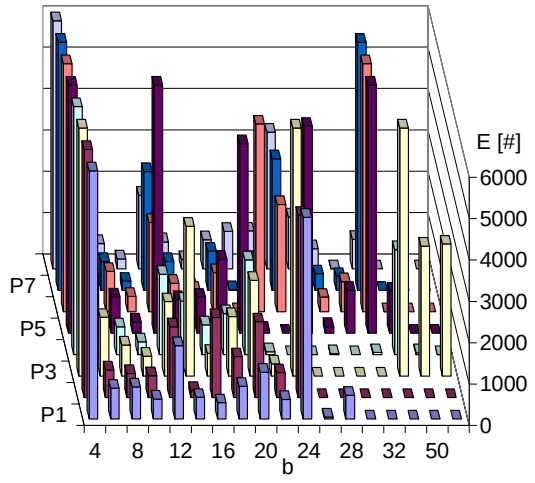


Figure 12. Number of expanded nodes E on the processors P_1 to P_8 versus the cube size b for the benchmark problem BOTTLENECK.

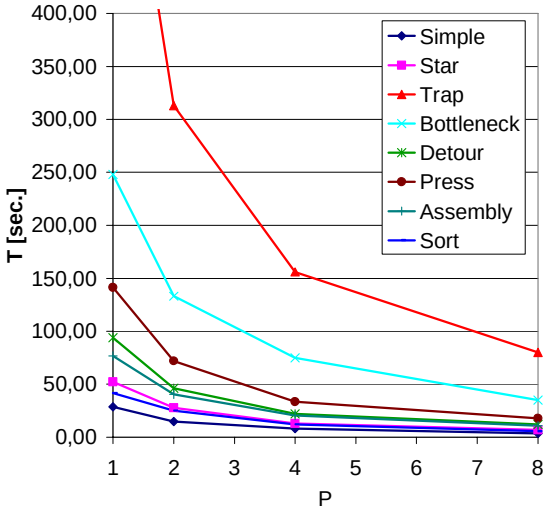


Figure 13. Parallel runtime T versus number P of processors for all benchmark and industrial problems.

3.2. Bidirectional Search

Path planning is a special type of search problem, where the start and the goal configuration are known in advance. Therefore, it is possible to search not only from the start to the goal (*forward search)*, but also from the goal to the start (*backward search)*. The *bidirectional search* performs both search directions simultaneously. The search task is finished as soon as the two search fronts meet each other. Bidirectional search offers two main advantages. First, the backward search can be much simpler than the forward search. Second, if the search fronts meet each other, the run-time can be reduced.

For implementing the bidirectional search, there are basically two ways: using one or two OPEN-lists. In the first way, the nodes of both search fronts are stored in a common OPEN-list. In each search iteration, the current best node is selected from this list regardless of which search direction it belongs to. This has the advantage that there is little additional effort. On the other hand, often only one search direction is pushed ahead. This is caused by the weight *w* of the *h* cost. The successor of the current best node usually has a better rating than the node itself, because it is located nearer to the goal. Thus, once a search direction is chosen, it

will hardly be changed again. As a disadvantage, the planning system may choose the wrong direction due to the uninformed heuristic. Finally, it is unlikely that the two search fronts will meet in the middle, thus the run-time improvement of the bidirectional search is lost. For the parallel version of the bidirectional search with domain decomposition, all these effects occur if parts of the two search fronts are located in hypercubes which are mapped on the same processor.

In the second way, two separate OPEN-lists are used for the forward and the backward search.[16] In each search iteration, the current best node is selected alternatively from the two lists processing both searches simultaneously. The overall run-time is at most twice the run-time of the fastest version of the forward and backward search processed separately. If the search fronts meet each other before finishing their task, the run-time decreases.

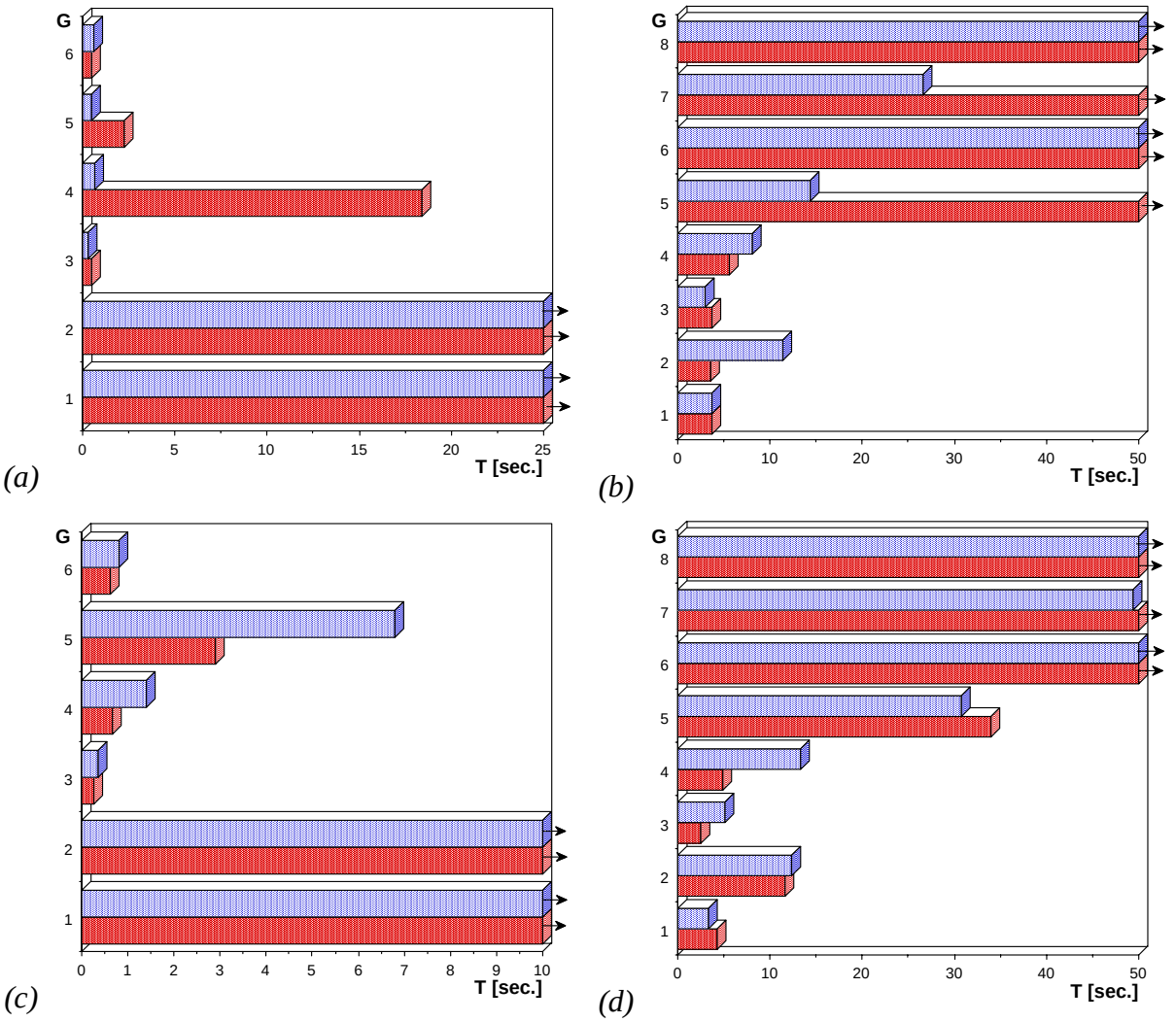


Figure 14. Run-times T *using the forward search to goal* G *(light gray) and using the backward search from goal* G *(dark gray) for benchmark problems SIMPLE (a) and STAR (b); run-time* T *for bidirectional search with one (dark gray) and two (light gray) OPEN-lists for different goals* G *of benchmark problems SIMPLE (c) and STAR (d).*

Experimental results with the run-times for the separate search for two benchmark problems are given in Figure 14 (a and b).[17] The forward direction of benchmark SIMPLE can usually be solved faster than the backward direction. For benchmark STAR, the results are mixed. For some search directions of both benchmark problems, no solution could be found by unidirectional search due to memory overflow (indicated by arrows in the figures). These results form the basis for the following comparisons concerning bidirectional search.

Experimental results for the bidirectional search with one and two OPEN-lists are shown in Figure 14 (c and d). For almost all goal configurations of both benchmark problems, the

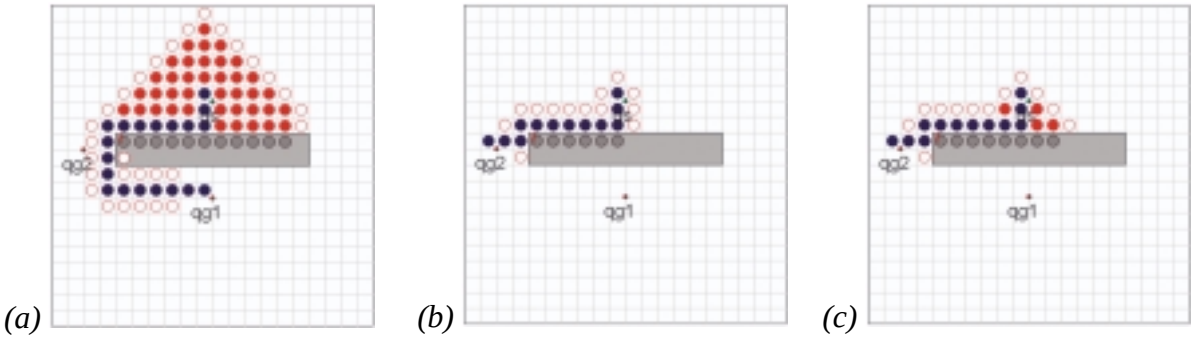
use of only one OPEN-list is faster than the use of two lists. The search was able to select the favorable search direction. As an exception, one OPEN-list for goal 7 of benchmark STAR fails because the planning method pushes the unfavorable direction. Here, two OPEN-lists are successful by simultaneously processing both directions. Compared with the unidirectional search in Figure 14 (a and b), the bidirectional search could solve one additional problem. The expected run-time reductions caused by meeting both search fronts could not be validated; this is certainly caused by the high weight *w* of the heuristic *h()*.

3.3. Dynamic Start and Goal Switching

Experiments have shown that the planning times are quite different for different start or goal configurations. They fluctuate between fractions of seconds up to the insolubility of the task. However, it is impossible for the user to recognize beforehand which start or goal is favorable and which is not. To provide the user with an automatic selection, the *goal switching* method can be applied.

Here, a single search is accomplished as in the original algorithm. If the planning system detects that another goal is more favorable while searching, it will switch its search direction to the new goal. Thus, the search always selects the current best goal. This switching can be implemented very simply by a small modification in the heuristic evaluation *h(n)* of the current node *n*: Instead of using the C-space distance of *n* to one goal, the minimum distance of *n* to all goals is used if it is smaller than the former one. With this, the search space is divided in different areas with nodes that are nearer to one goal than to all other goals.

In a C-space with no obstacles, goal switching will have no effect. Due to the best-first paradigm, the search will choose one goal and runs directly toward it. The other goals are no longer considered. The goal switching occurs first, if an obstacle blocks the direct way to the goal. In this case, the best-first search tries to surround the obstacle. During this operation, it can happen that a node lying in the area of another goal is expanded. The prior goal is dropped and the search switches to the new goal (Figure 15).



(a) *(b)* *(c)*

Figure 15. Example in the two-dimensional C-space with two goals: (a and b) planning from start q_S to the goals q_{G_1} and q_{G_2}, (c) planning with dynamic goal switching from one start to the two goals. The white, gray, and black dots represent nodes in OPEN, in CLOSED, and on the solution path, respectively.

Analogue to goal switching, the start switching can automatically select the simplest start and goal pair, if multiple start configurations exist. As the OPEN-list is by definition able to maintain multiple nodes, all start configurations are added to OPEN at the beginning. During the search it may happen that the next best node of OPEN belongs to a different start configuration, thus, a dynamic start switching takes place.[18]

4. Multi-Goal path planning

The combination of bidirectional search together with a dynamic start and goal switching leads to a MTP search solving the MTP path planning problem.[18] The *MTP path planning problem* is to find a collision-free path connecting a set of goal configurations minimizing some criterion function such as the total path length.

To solve this MTP path planning problem, every configuration represents a node in an initial graph (Figure 16a). In each iteration, one collision-free path between a set of start configurations and a set of goal configurations is computed. The start and goal configurations are specified by different selection strategies, which will be presented in Section 4.10. The solution path is then inserted as the corresponding edge in the graph (Figure 16b). In this iteratively growing *MTP graph* a shortest sequence planning mechanism tries to find the shortest sequence to solve the given MTP problem (Figure 16c). The shortest sequence planning will be presented in Section 4.2.

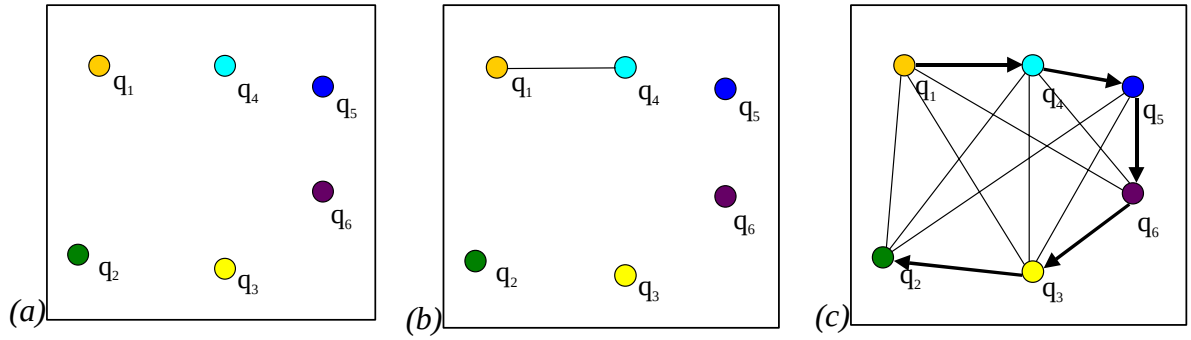


Figure 16. (a) Initial MTP graph.(b) Inserted solution edge (q_1, q_4) *after one call of the* MTP path planning, *(c) Solution sequence after 13 runs.*

In the worst case, the maximal number $R_{Max} = N * (N-1)/2$ of planning runs might be necessary for N configurations to find the optimal sequence. But by applying the MTP path planning the total number of runs as well as the total solving-time can be reduced significantly.

4.1. **Goal Selection Strategies**

Its main task consists of selecting the suitable start and goal configuration pair for the MTP path planning in every iteration. We have investigated random and deterministic selection strategies, which can be summarized as follows.

In the simplest case, the *randomized pair selection* P0, two not yet connected configurations q_i and q_j are randomly selected. As this method has no knowledge about the distance between the configurations, many unnecessary planning runs have to be made to find a valid sequence.

In contrast, the *nearest pair selection* strategy P1 selects the configuration pairs according to a specified order, e.g., the Euclidean distance between start and goal. In some MTP problems, an easy planning task (short distance at the beginning) may need a long time until a collision-free path can be found, because the direct way is blocked by an obstacle.

While P0 and P1 can be solved with any PTP path planning approach, the *nearest goal selection* P2 needs at least a dynamic goal switching. Beginning at the specified start configuration, the path planner receives all the other $N - 1$ configurations as possible goals. As the path planner will find the shortest path at first, the nearest configuration will be selected automatically. In the next run, the planner continues with the previous goal and tries to find a collision-free path to all remaining configurations. After $N - 1$ runs, a valid sequence is found.

If the path planner is additionally able to cope with multiple start configurations, thus selecting automatically the easiest start and goal pair, then the *adaptive pair selection* P3 can be applied. In this strategy, the planning system receives all configurations including a list of edges (representing the missing collision-free paths) which still have to be computed.

4.2. Shortest Sequence Planning

Finding the shortest sequence is similar to the traveling salesman problem (TSP).[19] There, the objective is to find an optimal tour through *n* towns, visiting each town at least once. Adapted to the MTP here, an industrial robot has to reach *N* configurations at least once. In contrast to the TSP, the robot rarely has to return to its starting pose. Therefore, we call this problem the *shortest sequence problem* (SSP).

The input of a TSP or SSP (xSP) is usually a graph with nodes and edges. In our application, this is the MTP graph. As a given pose may be reached by several different configurations, the graph contains groups of nodes. In this case, a valid sequence consists of one node of each group, in order to reach every pose at least once. The similar TSP scenario would provide different suburbs for each town, and the salesman has to visit only one suburb of every town. We call these extended problems TSP++ and SSP++.

For solving the sequence planning, the MTP graph must contain as many edges as possible. As the graph is iteratively growing, missing edges between node q_i and q_j via q_k can be added virtually if edges between q_i and q_k and q_k and q_j exist. Graph completing can simply be done by applying a standard "shortest-path-algorithm" as, e.g., the Dijkstra Algorithm[20] as soon as the MTP graph is connected.

While in the basic xSP cases the standard connectivity[19] test is sufficient, it has to be extended for xSP++. For these cases, the standard test can be applied, but a feasible xSP++ solution may not be recognized as soon as possible. In Figure 17 17 (a and b) two different examples are shown, each with one start configuration, two middle poses with three alternative configurations, and one goal configuration. After four planning runs, in both cases a standard connectivity test would fail, although the example in Figure 17b is "xSP++ connected."

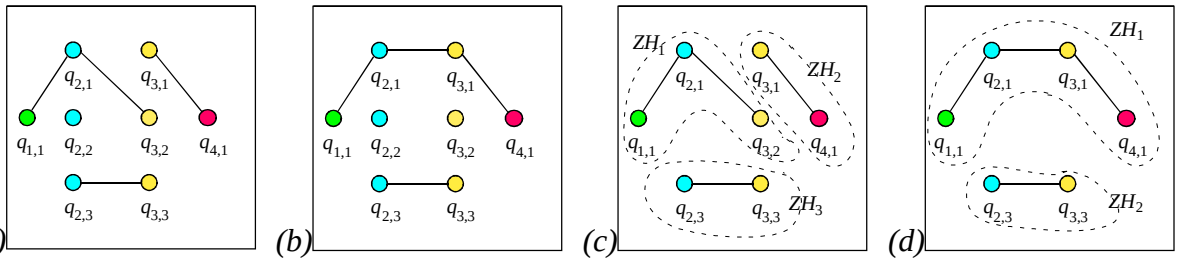


Figure 17. (a and b) MTP graph for two different MTP problems, (c and d) Connected components ZH_i.

To avoid this disadvantage, the xSP++ connectivity is checked in the following way: At first, a standard algorithm computes the connected components[19] ZH_i in the MTP graph. Then it will be tested to see whether one component contains (1) the start configuration, (2) at least one configuration of every pose , and (3) at least one configuration of the final pose. If one component fulfills these conditions, the MTP graph is called *xSP++ connected*.

The example in Figure 17a contains three connected components, but none of them fulfills the three conditions, thus this example is not xSP++ connected. In the second example, the connected component ZH_1 fulfills all three conditions, so this example is xSP++ connected.

Based on this completed graph, a xSP solver can find the shortest tour or sequence. While solution methods for TSPs have already been thoroughly investigated (see, e.g., 120 cities[21], 532 cities[22], 666 cities[23], and 13.509 cities[24]), no methods for xSP++ tasks have yet

been considered to our knowledge. By extending the available implementation of a TSP solver (Pederson[25]), we have developed a new xSP++ solver.

Pederson has used genetic algorithms (GA)[26] to solve the traveling salesman problem. Adapted to MTP problems, a *gene* stands for a goal configuration, a *chromosome* represents a valid sequence, and a *population* is a set of multiple chromosomes. Every chromosome in one population is rated by a *fitness evaluation* function, thus the best chromosome representing the best sequence can be found.

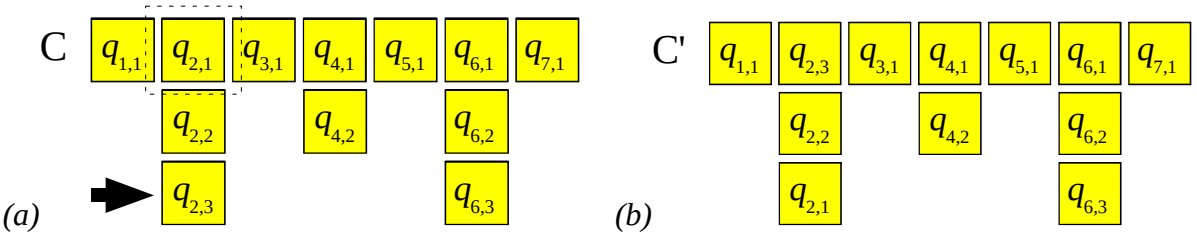


Figure 18. New exchange function for our genetic algorithm: (a) One gene is randomly selected (dashed rectangle). If the gene contains more than one allele, the currently selected allele is exchanged by another randomly selected allele (arrow). (b) The resulting chromosome.

A population can grow by applying the standard GA functions crossover and mutation. To handle also a group of nodes (multiple configurations of a TCP), the meaning of a gene must be extended. Here, a gene contains different values, denoted as alleles. As a chromosome represents a valid sequence, every gene selects one allele as the current member of the sequence.

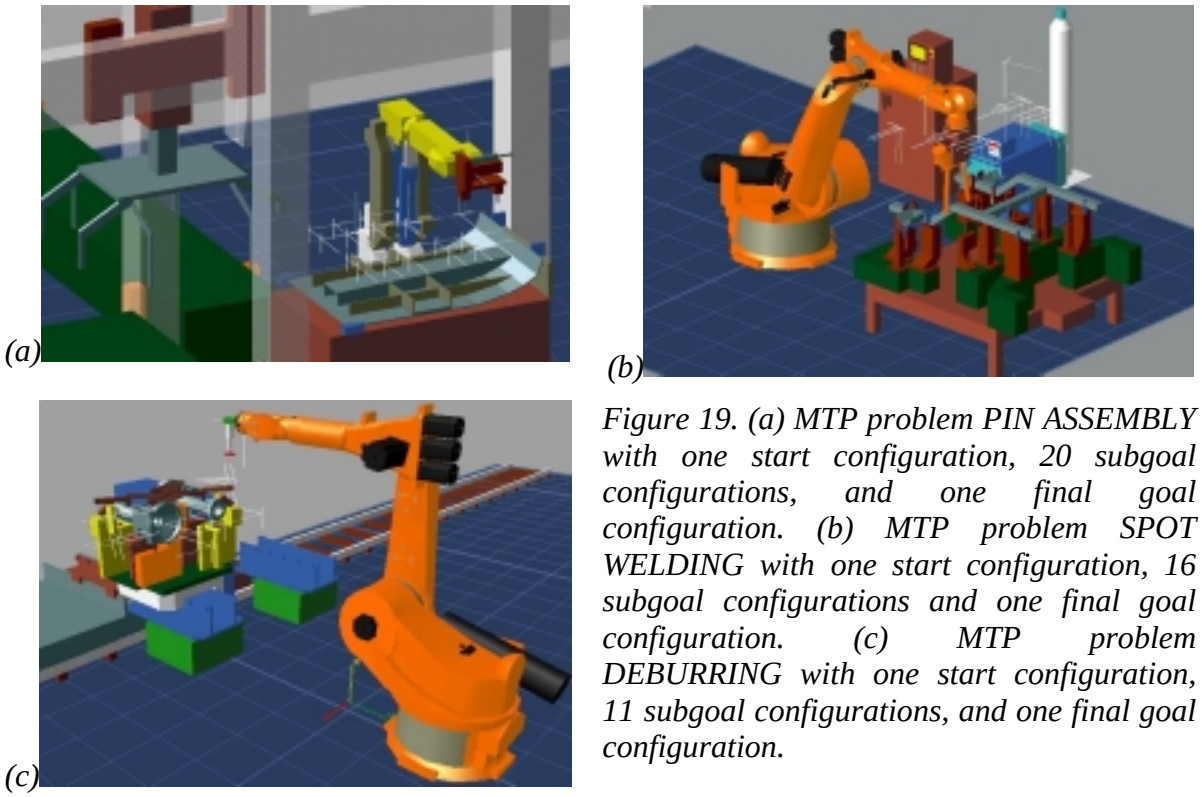


Figure 19. (a) MTP problem PIN ASSEMBLY with one start configuration, 20 subgoal configurations, and one final goal configuration. (b) MTP problem SPOT WELDING with one start configuration, 16 subgoal configurations and one final goal configuration. (c) MTP problem DEBURRING with one start configuration, 11 subgoal configurations, and one final goal configuration.

C*rossover* as well as *mutation* can therefore still be applied on these new chromosomes. The new function *exchange* was added to exchange a randomly selected allele of a randomly selected gene, thus modifying a chromosome. As this function is applied only on one chromosome, it is very similar to the mutation function and could of course be integrated in it (Figure 18).

For each problem, we have solved the maximum number of planning runs (PIN ASSEMBLY: 231, SPOT WELDING: 153, DEBURRING: 148) 10 times, to show the

averaged performance of the different strategies. In all problems, the randomized strategy P0 shows the worst results (Figure 20).

4.3. Experimental Results

We have implemented this MTP approach on a workstation (Pentium PC with 350 MHz processor and 128 MB memory) running under the LINUX operating system.[15] The MTP path planner is implemented in C language and runs as a server process.

The MTP control unit is written in C++ using LEDA[27] and communicates via the parallel virtual machine interface with the path planner. For comparing the four goal selection methods, we have tested the MTP approach on several industrial MTP problems (Figure 19).

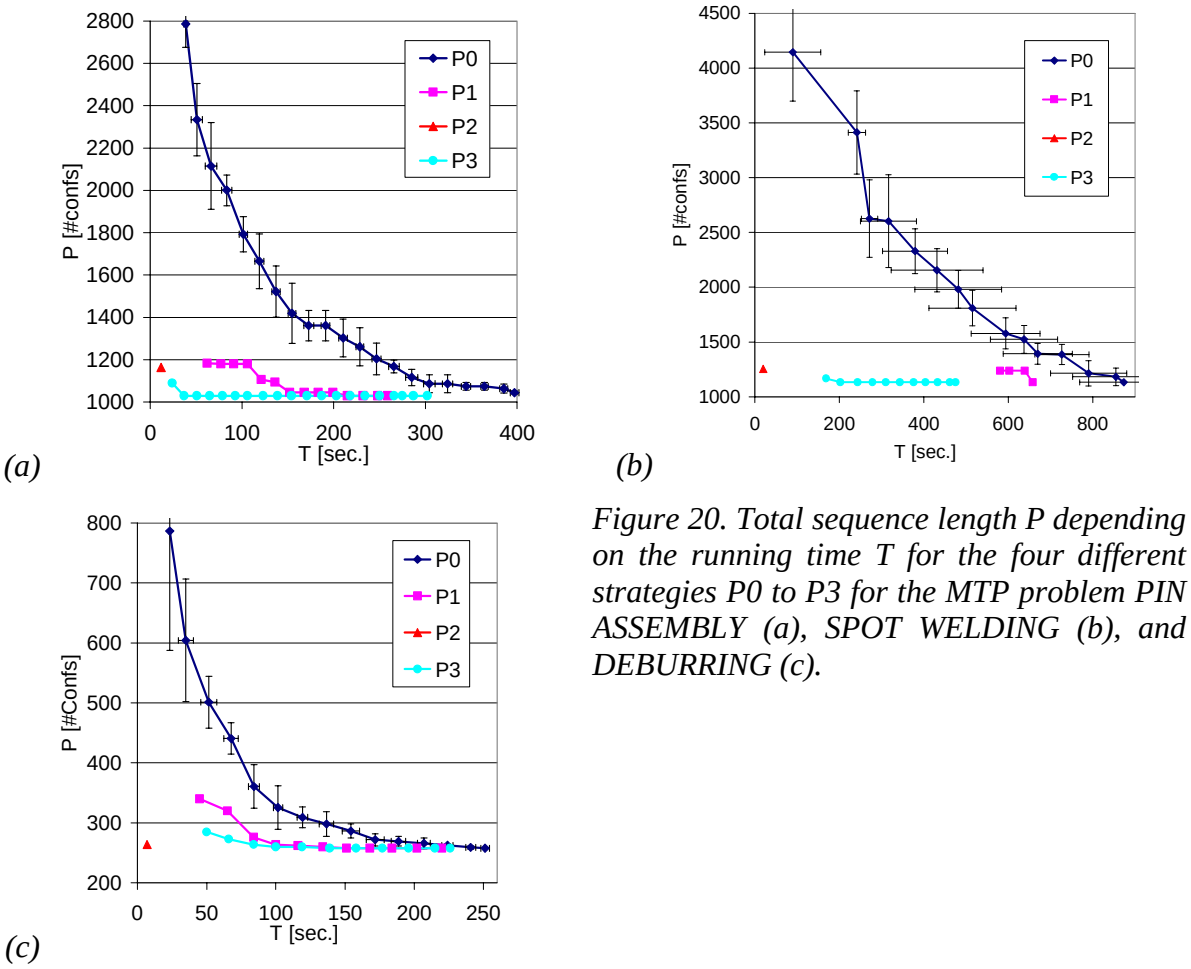


Figure 20. Total sequence length P depending on the running time T for the four different strategies P0 to P3 for the MTP problem PIN ASSEMBLY (a), SPOT WELDING (b), and DEBURRING (c).

5. Conclusion

With the help of a path planner the off-line programming of robots can be accelerated, thus reducing the setup cost of the capital intensive installations in production companies. Yet, path planning approaches are virtually nonexistent in industry. To the best of our knowledge only one such system exists.[28]

We have presented a planning approach for industrial robots with 6 DOF which works in the implicit and discretized C-space. Via the discretization, the search space size can be limited. Experiments have shown that all the benchmark problems we have considered can be solved with a size of $N < 10^{10}$. If the search space is larger (finer discretization), a collision-free path could of course be planned, but slight changes to the benchmark problem, e.g., different obstacle locations or modified start or goal configurations, soon lead to memory overflow. These results point out the great challenge of path planning approaches.

In contrast to other existing grid-based planning methods, we use distance computation for collision detection. On the one hand, the distance information can be buffered and efficiently used for the evaluation of neighboring configurations. Hence a lot of time consuming calculations can be saved. On the other hand, the distance can be used for hierarchical search, enabling the planner to make steps as large as possible (see, e.g., Henrich et al.[29] and Autere and Lehtinne[30]). In our future work we will concentrate on improving the hierarchical search.

To accelerate the basic approach, we have investigated several extensions. The parallelization with static load balancing results in a balanced load distribution and shows very good speedups. The main problem of introducing these promising results into industrial applications is the low industrial interest in investing in expensive robot simulation software and in parallel computing architectures. Thus, powerful graphic workstations with more than 4 processors are needed, which are supported by simulation systems. The bidirectional search leads at least to an automatic selection of the easier search direction, but a run-time reduction caused by a meeting of the search fronts in the middle could not be achieved. A wave shaping approach, which is difficult to integrate in a parallel search, may improve the results. Dynamic start and goal switching enable the planning system to automatically select a favorable start and goal pair. Thus it is no longer necessary that the user select the start or goal configurations.

Based on these extensions, we have finally developed the new multi-goal search to solve MTP problems. We have introduced four different goal selection methods and compared them for three industrial MTP problems. With P2 and P3 the simplest path segment are solved first. With every additional path planning, the total sequence path length can be further reduced, thus the MTP can be interrupted at any time after a first sequence is available. With all other existing path planners, the MTP problem can only be solved with random pair or nearest pair selection. To find the shortest sequences, a new GA based xSP++ solver and a modified MTP graph connectivity test have been developed.

For the future, we are focusing on developing a path planner which is able to cope with moving objects and gripped workpiece. With some modifications, our approach is also suitable for tasks in the area of virtual engineering. Instead of planning the path for robots, we are able to search a trajectory for the components which have to be assembled obtain the final object. Additionally, we are developing a path smoothing method for executing the computed trajectories with a real robot.

Acknowledgements

The main part of this work was performed within the project "Scalable algorithms for parallel motion planning in dynamic environments" funded by the German Basic Research Framework (DFG-Schwerpunktprogramm) "Efficient algorithms for discrete problems and their applications".

References

[1] H. Wörn, Ch. Wurll, and D. Henrich: Automatic off-line programming and motion planning for industrial robots, 29th Int Symp Robotics (ISR'98), 1998.

[2] Y. K. Hwang and N. Ahuja: Gross motion planning – A survey, ACM Comput Surv, 24(3), (1992).

[3] P. E. Hart, N. J. Nilsson, and B. Raphael: A formal basis for the heuristic determination of minimum cost paths, IEEE Trans Syst Sci and Cybernet, January 1968, pp. 100-107.

[4] K. Gupta, and A. P. del Pobil (Editors): Practical motion planning in robotics, Wiley, New York, 1998

[5] B. Baginski: Motion planning for manipulators with many degrees of freedom - the BB-method, Dissertation, Fakultät für Informatik, Technische Universität München, 1998.

[6] P. Chen and Y. K. Hwang: A dynamic graph search algorithm for motion planning, IEEE Trans Robotics Automat 14, (1998).

[7] K. H. Hwang: Completeness vs. efficiency in real applications of motion planning, IEEE Workshop on Practical Motion Planning Robotics, Minneapolis, April 1996.

[8] G. Katz: Integration eines parallelen Bewegungsplaners in ROBCAD, Master's Thesis, Institute for Process Control and Robotics, University of Karlsruhe, 1996.

[9] D. Henrich, Ch. Wurll, H. Wörn: On-line path planning with optimal C-space discretization, IEEE/RSJ Int Conf Intelli Robots Systems (IROS'98), Victoria, Canada, October 12-16, 1998.

[10] C. Qin and D. Henrich: Path planning for industrial robot arms - a parallel randomized approach, Int. Symp Intell Robotic Syst (SIRS´96), Lisabon, Portugal, July 22-26, 1996, pp. 65-72.

[11] D. Henrich, Ch.Wurll, and H. Wörn: 6 DOF path planning in dynamic environments – a parallel on-line approach. Proc IEEE Int Conf Robotics Automat (ICRA'98), Leuven, Belgium, May 16-21, 1998.

[12] D. Henrich and X. Cheng: Fast distance computation for on-line collision detection with multi-arm robots, IEEE Int Conf Robotics Automat, Nice, France, May 10-15, 1992, pp 2514-2519.

[13] D. Henrich, S. Gontermann, and H. Wörn: Kollisionserkennung durch parallele Abstandsberechnung, 13. Fachgespräch Autonome Mobile Syst (AMS'97), Stuttgart, October 6-7, 1997, Reihe "Informatik Aktuell," Springer-Verlag, Berlin.

[14] T. Osterroht: Segment-based distance computation for an on-line path planner, Studienarbeit, Institut für Prozeßrechentechnik, Automation und Robotik (IPR), Universität Karlsruhe (TH), 1999.

[15] C. Wurll and D. Henrich: Ein Workstation-Cluster für paralleles Rechnen in Robotik-Anwendungen, Proc 4. ITG/GI-Fachtagung Arbeitsplatz-Rechensyst (APS'97), Universität Koblenz-Landau, May 21-22, 1997, pp. 187-196.

[16] D. Nassimi, M. Joshi, and A. Sohn: H-PBS: A hash-based scalable technique for parallel bidirectional search, Proc 7th IEEE Symp PDP, San Antonio, 1995, pp. 414-421.

[17] D. Henrich, Ch.Wurll, and H. Wörn: Multi-directional search with goal switching for robot path planning, Lecture Notes in Computer Science 1416, (Editors), Tasks and methods in applied artificial intelligence, A. P. del Pobil, J. Mira, and M. Ali, Springer-Verlag ,Berlin, 1998, pp. 75-84.

[18] Ch. Wurll, D. Henrich, and H. Wörn: Multi-goal path planning for industrial robots, Robotics Applicat´99, October 1999.

[19] D. Jungnickel: Graphs, networks and algorithms, Springer-Verlag, Berlin, 1999.

[20] E.W. Dijkstra: A note on two problems in connexion with graphs, Numer. Math. (1959), 1269-271.

[21] M. Grötschel: On the symmetric travelling salesman problem: solution of a 120-city problem, Math Progr Stud 12 (1980) 61-77.

[22] M. Padberg, and G. Rinaldi: Optimization of a 532-city symmetric traveling salesman problem by branch and cut, Oper Res Lett 6 (1987)1-7.

[23] M. Grötschel, and O. Holland: Solution of large-scale symmetric traveling salesman problems, Math Program (1991) 141-202.

[24] D. Applegate, R. Bixby; V. Chvátaland, and W. Cook: On the solution of Traveling salesman problems, Doc Math J. DMV Extra Vol. ICM III, (1998, 645-656.

[25] Th. Pederson: GATSS – a genetic based solver of a traveling salesman problem, http://www.acc.umn.se/~top/travel_information.html, 1995.

[26] J.H. Holland: Adaption in natural and artificial systems, University of Michigan Press, Ann Arbor, 1975.

[27] K. Mehlhorn, and St. Näher: The LEDA platform of combinatorial and geometric computing, to appear (see http://www.mpi-sb.mpg.de/LEDA/).

[28] L. Overgaard, R. Larsen and N. Jacobsen: Industrial Applications of the AMROSE Motion Planner, Practical motion planning in robotics, K. Gupta and A.P. del Pobil (Editors), Wiley , New York, 1998, pp. 155-183.

[29] D. Henrich, Ch. Wurll, and H. Wörn: On-line path planning by heuristic hierarchical search, IECON'98: 24th Annual Conf IEEE Ind Electron Soc, 1998.

[30] A Autere, and J. Lehtinne: Robot motion Planning by a hierarchical search on a modified discretized configuration space, IEEE Int Conf Robotics and Syst (IROS´97), 1997.