PATH PLANNING BY MULTIHEURISTIC SEARCH VIA SUBGOALS

Pekka Isto
Laboratory of Information Processing Science
Helsinki University of Technology
Otakaari 1, FIN-02150, Espoo, Finland

ABSTRACT

An efficient path planning algorithm for general 6 degrees of freedom robots is presented in the paper. The path planner is based on multiheuristic A^* search algorithm with dynamic subgoal generation for rapid escaping from deep local-minimum wells. The algorithm has been implemented as an extension to a robot off-line programming and simulation system for testing. The presented test results demonstrate that the algorithm is practically applicable to path planning for devices of different kinematic structure.

INTRODUCTION

Computation of a collision-free path for an movable object among obstacles is an important problem in the fields of robotics, CIM and AI. Various automatic task level programming systems can be build for robot guidance, teleoperation, assembly and disassembly among others, if a suitable method for path planning is available. In this paper an algorithm for path planning is presented. It is an extension of the well known and widely applied A^* search algorithm (Hart et al. 1968). However, the A^* algorithm is very susceptible to local-minimum wells, that it has to "fill" in order to proceed in the search.

The proposed algorithm uses a bidirectional multiheuristic A^* based search engine as a local planner. If the search encounters a deep local-minimum well, a random subgoal is generated and the search is continued for the subpaths from the original start to the subgoal and from the subgoal to the original goal. This procedure is continued recursively, until a path is found.

THE PROBLEM OF PATH PLANNING

Path planning has been studied for decades now. For surveys, see (Latombe 1991; Hwang and Anhuja 1992). Although theoretical solutions to the problem have been presented, for example (Canny 1988), only a few practical path planning algorithms have emerged, for example (Barraquand and Latombe 1991; Chen and Hwang 1992).

Most of the current approaches to path planning are based on the concept of configuration space (*C-space*) introduced by Lozano-Pérez and Wesley (Lozano-Pérez and Wesley 1979). *C-space* is the set of all possible configurations of a robot. The number of independent parameters needed to fully specify a robot configuration is the dimension of the *C-space*. The path planning problem has been proven to be a PSPACE-hard problem (Reif 1979). The high dimensionality of the *C-space* is seen as the principal reason for the difficultiness of the problem (Hwang and Anhuja 1992).

GRAPH SEARCHING APPROACH

One approach to path planning is to compute a decomposition of the *C-space* and search the graph connecting collision-free areas of the decomposition for a collision-free sequence. Unfortunately, the size of the *C-space* makes it impossible to precompute the decomposition for devices of more than 4 degrees of freedom (*DOF*'s). An alternative to the precomputation is to use heuristics to guide the search and to compute the decomposition as the search proceeds.

The proposed algorithm can be characterized as a heuristic search in the graph formed by discretizing the *C-space* into a rectangular grid. The graph representation is calculated as the search front proceeds away from the start configuration. The graph is computed only for a heuristically selected part of the *C-space*.

The problem with using grid searching techniques for path planning is the existence of local-minimum wells in the search space (*C-space*) (Latombe 1991). An informal definition for a local-minimum well is a region of the *C-space* where the heuristic function guides the search to a dead end. A^* and many other search algorithms have to fill up the wells in order to escape from them. This means generating and checking for collisions all of the configurations residing in the well. The resulting computational cost makes these simple algorithms impractical for difficult path planning problems or for devices with high number of degrees of freedom. Hence, many different techniques have been developed for escaping from the local-minimum wells i.e. (Barraquand and Latombe 1991; Pal and Jayarajan 1993).

THE SEARCH ALGORITHM

The search engine of the path planner is an improved version of the free-space expansion algorithm developed by Kondo (Kondo 1991). The algorithm performs staged multiheuristic search on the grid representation of the *C-space*. The search is A^* based, but instead of the usual one heuristic function the search is guided by a set of heuristic functions. These functions are allowed to guide the search sequentially in a round-robin fashion. The number of node expansions allocated for each function is calculated during the search based on the relative efficiency of the function for guiding the search away from the start configuration.

Using several heuristic functions is also a method to speed up the escape from the local-minimum wells. The heuristic functions utilized in the algorithm have different preferred search directions in the *C-space*. Since the more efficient heuristic functions guide the search more than the less efficient ones, the filling of the wells proceeds to the shallowest direction and the apparent sizes of the local-minimum wells are reduced.

Because a collision check is a computationally expensive operation, the improved search algorithm has been designed to use lazy evaluation principle to decrease the number of collision checks performed during the search. First, the nodes are checked for collision when they are taken from the OPEN set for expansion, not when they are generated. Since up to $2 * DOF$ new nodes are generated when expanding a node, this allows a substantial decrease in the number of performed collision checks. Only the collision-free nodes are expanded, the non-free space nodes are just inserted into the CLOSED set. Second, the collision status of a node is stored in its representation. This allows saving the redundant checking of the reopened nodes. It was experimentally found out that up to a third of the *open* operations can be reopenings, so the slight increase in the memory usage pays off with a decrease in the number of performed collision checks. Similarly, according to lazy evaluation principle, the reopened nodes are just re-evaluated and inserted back to the OPEN set. An alternative is to propagate the better path to the successor nodes as described in (Rich and Knight 1991). However, all this effort is wasted, if the successors were not selected for expansion in the later stages of the search.

The evaluation functions are similar to those used by Kondo, but with some important differences. Since the opened nodes are the ones selected by the heuristics, they describe it better than the expanded nodes. Therefore, the calculation of the evaluation values is based on the opened collision-free nodes instead of generated collision-free nodes.

After the *j* th stage, an evaluation value

$$P_t(j) = \frac{\sum_{(for\ \ last\ \ Q\ \ nodes)} p_t(C)}{Q}$$

is calculated for each heuristics $t = 1,...,T$ where T is the number of heuristics utilized and $Q=20$. The value of $p_t(C)$ is calculated for each opened node C by the following equation:

$$p_t(C) = \frac{g(C)^{DOF}}{F_t(C)},$$

where $g(C)$ is the distance from the start node to the current node C in grid steps and $F_t(C)$ is the total number of nodes opened by the *t* th heuristics until the expansion of node C.

For the first stage, each heuristics is allocated $E_{init} = 25$ node expansions. For the subsequent stages each heuristics is allocated

$$E_t(j+1) = \max\left[E_{init}\frac{P_t(j)}{\max\left[P_1(j),...,P_T(j)\right]},1\right]$$

node expansions.

For the full utilization of the multiple heuristics approach, it is essential to have the maximum operation in the previous equation. It keeps heuristics from being "zeroed out" by guaranteeing that each heuristics will perform at least one node expansion in each stage. Since every expanded node is evaluated by each of the heuristics and inserted to the common OPEN set, the heuristic functions are co-operating in the sense that they generate nodes for each other. The nodes generated by one heuristics may cause one of the other heuristics to jump out of a local-minimum well. For example, a heuristics bending the wrist of a manipulator may enable the heuristics rotating the base joint of the manipulator to continue motion towards the goal configuration.

Also a second evaluation value is calculated for each opened node according to the equation:

$$O_t(C) = \frac{F_t(C)}{g(C)}.$$

If the evaluation value $O_t(C)$ for the *t* th heuristic function decreases below a threshold value O_{th}, the execution of the heuristic is discontinued for that stage. Since the heuristic functions are co-operating, the discontinued heuristic may become active again in some later stage of the search. If all of the heuristics are discontinued, the search has encountered a serious local-minimum well and a subgoal is generated as described below. The value for O_{th} is a user defined parameter.

Since the search direction can have a dramatic effect on the search efficiency, the search engine is bidirectional. The bidirectionality is implemented by executing unidirectional search from both directions.

In accordance to Pohl's cardinality comparison principle (Pohl 1971), the size of the OPEN set is used to select the search direction. After completing a search stage, the search direction with the smaller OPEN set is selected for the next stage. This choice makes the search algorithm select the more efficient search direction from the more cluttered space to the open space. This takes place because there are more collisions in the cluttered space. As the non-free space nodes are not expanded, the size of the OPEN set grows slower allowing more search effort being spent in that particular search direction.

A multiheuristic A^* search can still easily get stuck in the local-minimum wells. Although it will escape them faster than a pure A^* search, it can still take hours to solve the planning task or it may fail to solve the task due to the amount of memory needed. Thus, multiple heuristics as such is not a sufficient technique for a practical path planner.

If a deep local-minimum well is encountered and all heuristics are discontinued, a detour is attempted. A random collision-free subgoal configuration is generated and the OPEN sets are sorted to reflect the distance to the subgoal. The search is then continued for the subpaths from the original start to the subgoal and from the subgoal to the original goal. Often, the detour avoids deep local-minimum wells, but if one is encountered again, the subgoal generation is continued recursively until a path avoiding deep local-minimum wells is found.

HEURISTICS

The evaluation of a node is performed with the two part function *f(C) = g(C) + h(C)*, where *g(C)* was given above and *h(C)* is the estimated cost from the current node to the goal node.

The heuristic functions *h(C)* utilized by the algorithm presented here are weighted manhattan distances between the evaluated configuration and the goal configuration. The choice of manhattan distance instead of euclidean distance is based on the expansion strategy of the algorithm. Nodes are expanded along the axes of the grid. Diagonal movements from one grid point to another are not allowed. Thus, manhattan distance is a better estimate of the distance to the goal and can be expected to yield better results than euclidean distance.

A problem with the approach taken by Kondo is that the heuristics for his preferred search strategy are randomly chosen. While he can demonstrate that randomly selected weight sets can yield good search efficiency when averaged over many sets, the efficiency for a particular run can be very low. Furthermore, variance in the search efficiency is an undesirable feature. A fixed set of weights producing good search efficiency would be a desirable improvement.

A reasonable set of fixed weights can be based on the following categorization of the devices and their degrees of freedom. The movement of the first joints of a manipulator-like device affects the position of all of the subsequent joints of the device. Therefore, the joints closer to the base of the device should be given higher weights than the joints further away from the base. The degrees of freedom of many devices can be partitioned into two sets. One set determines the position of the device and the other set determines the orientation of the device. This model suits for example a rigid body. If there is no obvious preference to pick, even weights for each degree of freedom is a reasonable choice. These qualitative sets can be defined more precisely as parameters for the heuristic function *h(C)*:

Manipulator heuristics:

$$a_i = \begin{cases} 9 - i, & i = 1,\ldots,7 \\ 1, & i = 8,\ldots,DOF \end{cases}$$

Position heuristics:

$$a_i = \begin{cases} 9, & i = 1,\ldots,d \\ 1, & i = d+1,\ldots,DOF \end{cases}$$

Rotation heuristics:

$$a_i = \begin{cases} 1, & i = 1,\ldots,d \\ 9, & i = d+1,\ldots,DOF \end{cases}$$

Even heuristics:

$$a_i = 5, \quad i = 1,\ldots DOF.$$

Above $d = \lfloor (DOF + 0.5)/ 2 \rfloor$. These are the four fixed weight sets used by the search engine.

Random subgoal generation often produces very poor solution paths in terms of path length and geometry. Similar problems have plagued also some of the previous path planners and the authors have developed simple but fairly effective geometrical optimizing algorithms (Barraquand and Latombe 1991; Berchtold and Glavina. 1994). These optimizers can be used for improving the quality of the solutions produced by the path planning algorithm presented in this paper. The simple optimizers try to repeatedly delete those vertices from the solution path that are not necessary to keep the path collision-free. These optimizers have a run-time proportional to the number of vertices. Therefore, if two configurations have the same value of *f(C)*, the expansion is continued to the same direction in the *C-space*. This will reduce the number of vertices in the solution. A momentum term ρ is used for tie-breaking. The momentum term has a value of 0.5, if the evaluated node was expanded in the same direction as the parent node along the axis *j*, or 0 otherwise.

Randomized weight-set generation was also implemented for comparison with the fixed heuristics. A randomized heuristics is produced by randomly selecting a value between 1 and 9 inclusive to be used as a weight a_j in the heuristic function. In the test cases, four such heuristics were generated for each run.

The full expression for the estimate of the cost to reach the goal cell is given in the equation:

$$h(C) = A\left(\sum_{i=1}^{DOF} a_i D_i(C,G) - \rho a_j \right),$$

where $D_i(C,G)$ is the distance between the current node and the goal node G in grid steps along axis i. The value for A is 3.

DATA STRUCTURES

The efficiency of the path planning algorithm can be qualified based on the number of performed collision checks. However, it is not enough for evaluating the performance of an algorithm. The free-space enumeration type of path planning algorithms, like the one presented in this paper, have the drawback of having to represent a portion of the high-dimensional configuration space. This incurs a considerable computational cost for this type of path planning algorithms.

Since the efficiency of the search algorithm implementation is directly affected by the efficiency of the underlying data structures, the issue of search space representation can make or break the practical usability of a particular algorithm. Thus, the representation of the configuration space for the path planning algorithm is a non-trivial problem.

In the current implementation of the path planner, the configuration space is represented as a DOF-dimensional grid of points. Each point represents a neighborhood in the *C-space*, which means that any configuration in the configuration space is equivalent to the nearest point in the grid.

There are two distinct types of functionality required from the data structures for the presented search algorithm. An inclusion test has to exist for checking the inclusion of a certain node in the OPEN-set or in the CLOSED-set. Also, there has to be a method to obtain a node with the lowest $f(C)$ value from the OPEN-set.

The inclusion test is implemented by making the sets resemble sparse matrixes. The discrete positions of the degrees of freedom are enumerated and the enumerations are used as vectors to identify the points in the grid and to index the sparse matrix. Only those points of the configuration space are stored in the sparse matrix (set), that are examined during the search process. If each dimension of the configuration space corresponds to a dimension in the sparse matrix, one can find a node by accessing the sparse matrix element, that has the same coordinates as the node's discretation point.

In the DOF-dimensional sparse matrix implementation, one has to access up to DOF rows to find an element. Since each row may contain only as many nodes as there are discrete positions for the corresponding degree of freedom, the access time for a node has an upper limit of $O(D_{max})$, where D_{max} is the maximum number of discrete positions. It should be noted that the N-dimensional sparse matrix data structure can be modified for an $O(log(D_{max}))$ access time by implementing the rows as balanced binary trees. At the moment, this modification has not been implemented, since it is doubtful that the improvement in the total search time will pay off the required programming effort.

For rapid access of the best node from a set, the nodes are sorted on $f(C)$. The nodes with a certain value of $f(C)$ are all bundled together and put into a bucket. The order of the buckets is maintained with a skip list (Pugh 1990). A skip list has a probabilistic $O(log(N_f))$ access time, where N_f is the number of different $f(C)$ values in the set. As the sets are a combination of a DOF-dimensional sparse matrix and a skip list, the set operations execute in $O(L_{max}+log(N_f))$, where L_{max} is the length of the longest row in the sparse matrix.

TEST CASES

The test cases for the presented algorithm can be divided into two categories. Several relatively easy problem cases are used to demonstrate that the predefined set of heuristics has a satisfactory performance. More difficult problems are used for demonstrating that the subgoal generation technique makes the algorithm powerful enough for practical use.

All reported times are CPU times on a SGI Indigo^2 computer with a 250Mhz R4400 processor and 128 megabytes of main memory. The object modeling and collision testing were done with TELEGRIP robot simulation and off-line programming system (Deneb Robotics Inc. 1994). Each degree of freedom of the devices was quantized into 100 discrete positions. The statistical data was calculated for at least 500 runs.

None of the single heuristics above could solve every task in the table 1 in less than 100000 collision checks, while search with the multiple fixed heuristics found a solution to every easy task in less than 3000 collision checks. It should be noted that the performance of the fixed heuristics is better than the average performance of the randomized heuristics. All of the easy problem cases were solved in 5 seconds by the multiple fixed heuristics based path planner.

Neither the fixed nor the randomized multiheuristic search algorithm could solve the difficult tasks of table 2 in 200000 collision checks. Those tasks are only solvable by the presented path planner with the help of the subgoal generation. The value for O_{th} was 50 for the SCARA of figure 2 and 60 for the rigid body of figure 3.

CONCLUSIONS AND FUTURE WORK

The combination of multiple heuristics and dynamic subgoal generation yields to several good properties in the algorithm. Easy planning tasks are solved rapidly by the multiheuristic search engine. Since the search engine is based on A^* algorithm and manhattan distances, the quality of the solutions found for easy problem cases is good if compared with the paths produced by many other algorithms. The subgoal generation mechanism allows the algorithm to rapidly escape deep local-minimum wells. Therefore, the algorithm is practically applicable to difficult problem cases too.

The search is performed in the configuration space of the device. Thus, the algorithm is applicable to a wide range of path planning tasks, as demonstrated by the test cases.

Two immediate problems with the presented algorithm are non-completeness and uncontrolled use of memory. However, both problems can be solved. Resolution completeness is lost because all of the generated subgoals have to become parts of the solution. An extension of the algorithm to build a subgoal graph (Faverjon and Tournassoud 1987) would allow the local search algorithm to connect the start, subgoal and goal configurations in any order until a solution is found. This subgoal graph approach should also improve the performance of the path planning algorithm, since the most difficult to reach subgoals can be left out from the solution.

A^* fully expands every opened node. This means that there will be $2*DOF$ new nodes per each opened node. Furthermore, there is no mechanism in A^* to limit the growth of the OPEN and CLOSED sets. As search algorithms are an active field of research, newer algorithms have been developed that perform search in restricted amount of memory and perform partial node expansion in order to limit the growth of the OPEN and CLOSED sets (Russell 1992). Such a search algorithm would relieve the potentially excessive memory need of the current path planning algorithm.

A version of the path planner will be integrated to the NEUROBOT car disassembly cell for on-line generation of robot movements (Tuominen et. al 1995). The NEUROBOT sensory system is used for obtaining the target locations for the various disassembly operations. The path planner is used for generating collision-free movements that can be used for constructing individual robot programs for each car to be disassembled.

REFERENCES

Barraquand, J.and J.C.Latombe. 1991. "Robot Motion Planning: A Distributed Representation Approach." The International Journal of Robotics Research 10, no. 6 (Dec.): 628-649.

Berchtold, S. and B.Glavina. 1994. "Scalable Optimizer for Automatically Generated Manipulator Motions." In Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, IEEE, Piscataway, NJ, 1796-1802.

Deneb Robotics Inc. 1994. TELEGRIP User Manual.

Canny, J. 1988. The Complexity of Robot Motion Planning, MIT Press, Cambridge, Mass.

Chen, P.C. and Y.K. Hwang. 1992. "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty." In Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, Los Alamitos, CA, 2346-2353.

Faverjon, B. and P.Tournassoud. 1987. "A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom." In Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, Washington, 1152-1159.

Hart, P.; N.Nilson and B.Raphael. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." IEEE Transactions on Systems Science and Cybernetics 4, no. 2 (Jul.): 100-107.

Hwang, Y. K.and N.Anhuja. 1992. "Gross Motion Planning - A Survey." ACM Computing Surveys 24, no. 3 (Sep.): 219-291.

Kondo, K. 1991. "Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration." IEEE Transactions on Robotics and Automation 7, no. 3 (Jun.): 267-277.

Latombe, J.C. 1991. Robot Motion Planning. Kluwer Academic Publishers, Norwell, Mass.

Lozano-Pérez, T. and M.Wesley. 1979. "An Algorithm for Planning Collision-free Paths among Polyhedral Obstacles." Communications of the ACM 22, no. 10 (Oct.): 560-570.

Pal, P.K. and K.Jayarajan. 1993. "Fast Path Planning for Robot Manipulators Using Spatial Relations in the Configuration Space." In Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, Los Alamitos, 668-673.

Pugh W. 1990. "Skip Lists: A Probabilistic Alternative to Balanced Trees." Communications of the ACM 33, no. 6 (Jun): 668-676.

Pohl, I. 1971. "Bi-directional Search." Machine Intelligence 6, Edinburgh University Press, Edinburgh, 127-140.

Reif, J.H. 1979. Complexity of the Mover's Problem and Generalizations." In Proceedings of the 20th IEEE Symposium on Foundations of Computer Science, IEEE, New York, 421-427.

Rich, E. and K.Knight. 1991. Artificial Intelligence, International Edition, McGraw-Hill, Singapore.

Russell, S. 1992. "Efficient Memory-Bounded Search Methods." In Proceedings of the 10th European Conference on Artificial Intelligence, John Wiley & Sons, Chichester, 1-5.

Tuominen, J; A.Autere; U.Berger and I.R.Meier. 1995. "Autonomous Robot Based Disassembly of Automotive Components." In Proceedings of the Conference on Integration in Manufacturing, IOS Press, Amsterdam, 341-352.

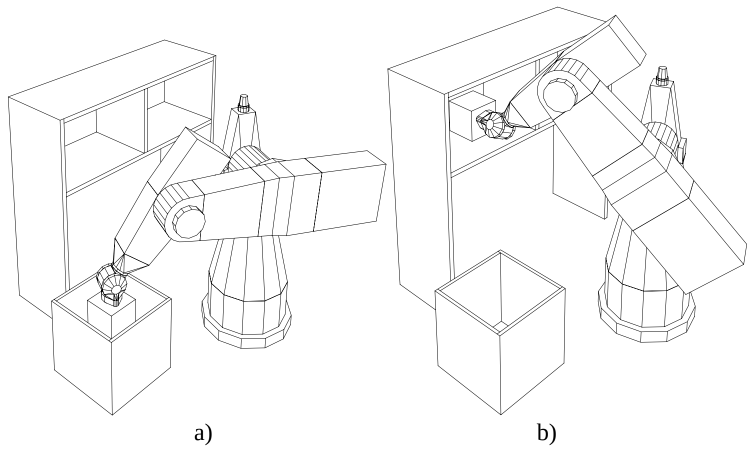

Figure 1: The start and goal configurations for a 6 DOF Puma robot.

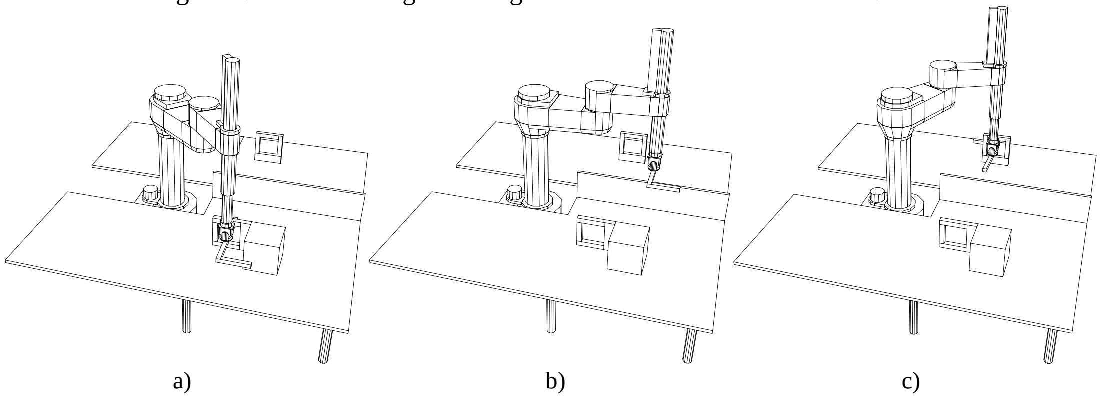

Figure 2: The start configuration and two goal configurations for a 5 DOF SCARA robot.

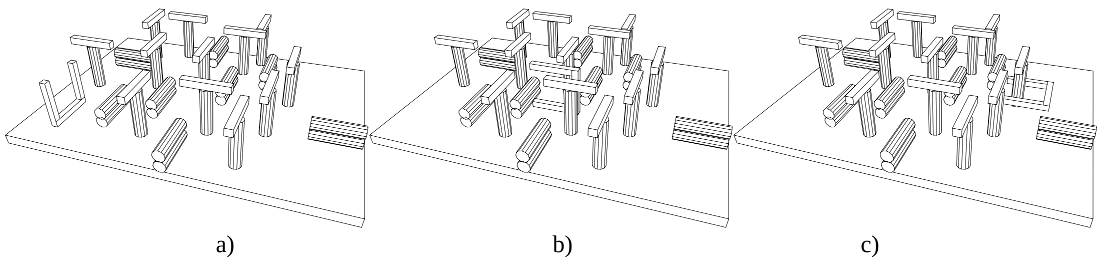

Figure 3: The start configuration and two goal configurations for a 6 DOF U-shaped rigid body.

Task	Manipulator	Position	Rotation	Even	Multiple fixed	Multiple randomized			
						ave	min	max	std
1a→1b	202355	27104	710	1764	909	7092	276	262664	20599
2a→2b	26190	>376384	418	7920	880	2255	312	42066	3456
2b→2c	3149	>330338	4463	3822	2640	4489	643	129605	8074
3a→3b	>403244	109306	100180	>460623	539	101800	302	397498	116410

Table 1: Results for the various heuristics. Numbers in the first column refer to figures above.

Task	Multiple fixed heuristics							
	Collision checks				CPU time			
	min	ave	max	std	min	ave	max	std
2a→2c	21840	79714	432916	70739	48.6	212.2	1340	214.2
3a→3c	90507	122960	359029	49202	229.9	331.2	1035	158.7

Table 2: Results for the difficult problem cases for the multiple fixed heuristics path planner with subgoal generation. CPU time in seconds.