*In: The 11th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Castellan, Spain, 1.-4. July, 1998.*

# Multi-directional search with goal switching for robot path planning

Dominik HENRICH, Christian WURLL, and Heinz WÖRN

Institute for Process Control and Robotics (IPR),
Computer Science Department, University of Karlsruhe,
P.O. Box 6980, D-76128 Karlsruhe, Germany,
E-mail: [dHenrich, Wurll, Woern]@ira.uka.de,
Http://wwwipr.ira.uka.de/~paro/

**Abstract**: We present a parallel path planning method that is able to automatically handle multiple goal configurations as input. There are two basic approaches, goal switching and bi-directional search, which are combined in the end. Goal switching dynamically selects a favourite goal depending on some distance function. The bi-directional search supports the backward search direction from the goal to the start configuration, which is probably faster. The multi-directional search with goal switching combines the advantages of goal switching and bi-directional search. Altogether, the planning system is enabled to select one of the preferable goal configuration by itself. All concepts are experimentally validated for a set of benchmark problems consisting of an industrial robot arm with six degrees of freedom in a 3D environment.

Keywords: path planning, graph search, bi-directional search, industrial robots, parallel processing

## 1 Introduction

The issue of robot path planning has been studied for a couple of decades and many important contributions to the problem have been made [Hwang92]. Path planning algorithms are of great theoretical interest, but are rarely used in practice because of their computational complexity [Kamal96]. Here, we take a step in the direction of practical path planning.

The problem of path planning that we focus on is the following: Input is the geometry of an industrial robot manipulator with six (rotational) degrees of freedom in an environment with static obstacles given by their current location. Optionally, there may be a couple of dynamic obstacles. All geometric objects are represented by a number of 3-dimensional convex[1] polyhedrons. Additionally, the start configuration of the robot and the goal position of the tool centre point (TCP) are given. The output of the problem is a sequence of pair-wise neighbouring and collision-free robot configurations from the start to one of the goals.

Up to now, the path planning systems search for a path between a given start and *one* given goal configuration. In many applications, there are *multiple* goal configurations available for path planning. These emerge because of the ambiguous solution of the inverse kinematics of most industrial robots, which may result in several goal configurations for the desired Cartesian TCP (see Figure 1). In this case, nowadays, the user has to select one of the goals for planning by hand. In this paper, we present a method that enables the path planning system to automatically exploit the available multiple goal configurations.

---

[1] This is no severe restriction, because every non-convex polyhedron can be modelled by (multiple) convex polyhedrons.

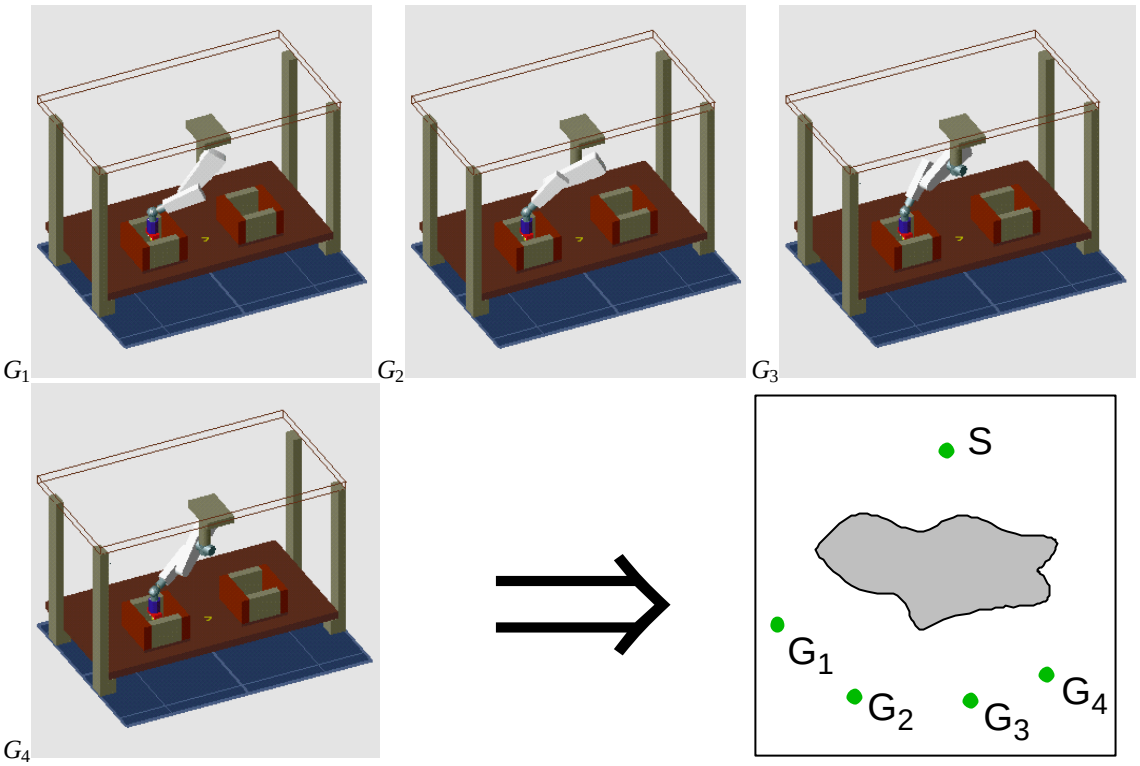


Figure 1: Multiple goal configurations $G_1$, ..., $G_4$ for one endeffect (TCP) position and an illustration of the resulting path planning problem with start configuration $S$. (Benchmark STAR from [Katz96])

The paper is organized as follows: In Section 2, we shortly review our basic approach to practical path planning. Then, we present two approaches which make use of multiple goal configurations. These are goal switching in Section 3 and bi-directional search in Section 4. Finally, we investigate the combination of both approaches called multi-directional search in Section 5. For each approach, we provide experimental results using a set of benchmark problems.

## 2 Basic approach

In order to avoid time consuming obstacle transformations into the robot configuration space (*C-space*), one can search in an implicitly represented C-space and detect collisions in the Cartesian workspace. This strategy enables the planning system to cope with on-line provided environments or moving obstacles, and to work reasonably fast in interactive robot programming environments (see Figure 2).

For searching in the implicit C-space, we apply the well-known A*-search algorithm[2] [Hart68, Korf92]. Therefore, the C-space is discretized and all robot configurations are represented by nodes building up the search space. The A*-algorithm maintains a CLOSED list of those nodes that have been expanded, and an OPEN list of those nodes that have been generated but not yet expanded. The algorithm begins with the start node on the OPEN list.[3] At each iteration, a node on the OPEN list with minimum heuristic evaluation is expanded, generating all of its children, and is placed on the CLOSED list.

[2] The results of this paper are not restricted to A*. They can be applied to other best-first search methods, too.

[3] OPEN and CLOSED can be implemented efficiently as priority list and hash table, respectively.

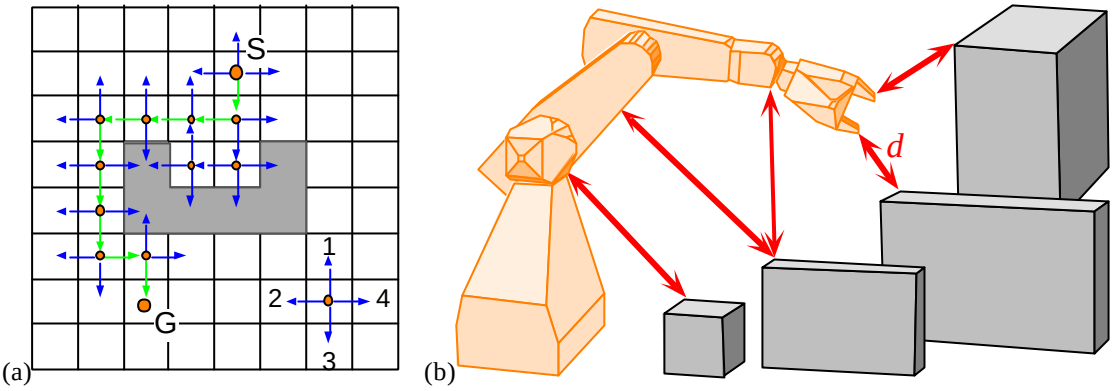


Figure 2: (a) A* search in the implicit C-space from the start configuration *S* to the goal configuration *G*. (The dots indicate investigated configurations and the arrows give reference to the corresponding successors.) (b) Collision detection in the explicit workspace by computing the minimum distance *d* between robot and obstacles

The evaluation function is applied to the children, and they are placed on the OPEN list in order of their heuristic values. The search continues until a goal node is chosen for expansion, or the OPEN list is empty. In the latter case, the algorithm stops with no solution. An illustration of the algorithm is shown in Figure 2a.

The evaluation function $f(n) = (1-w)g(n) + wh(n)$ is used, where $g(n)$ is the number of nodes of the path from the start node to node $n$, and $h(n)$ is the Airplane distance in C-space between node $n$ to the goal node $G_i$.[4] In the basic approach, the user has to select the goal configuration $G_i$. Collisions are detected by a fast, hierarchical distance computation in the 3D workspace, based on the given CAD model of the environment and the robot [Henrich92, Henrich97e] (see Figure 2b).

For parallelizing[5] the A*-algorithm, the configurations in OPEN and CLOSED must be accessible to all processors in order to distribute the work. These lists can either be managed by one dedicated processor or each processor has its own local lists. In commonly available message passing systems, each accessing of a global list results in a high communication effort. Thus, the local method is preferred. The work distribution is the key aspect of parallelization. Here, the C-space is decomposed into $d$-dimensional hypercubes of size $b$ in each dimension. For parallel processing, the hypercubes are cyclically mapped blockwise on the $p$ available processors. For details, see [Wurll98a].

We have implemented the parallel path planning method on a workstation cluster. The cluster consists of 9 PCs, each with 133 Mhz Intel Pentium processors and 64 Mbyte memory. The parallel communication is established by an Ethernet based bus network. For more details see [Wurll97a]. For testing the path planning method, we have developed five benchmark problems for the 6 DOF robot Puma260 [Katz96][6]. Especially benchmark problems STAR (see Figure 1) and SIMPLE are interesting, since these have multiple goal configurations. All other benchmarks have only one goal configuration because the TCP is located in a rather restricted area. Thus, the run-times of the following extended approaches are similar to the one of the basic approach.

---

[4] 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.

[5] An overview and classification of different approaches to parallel motion planning can be found in [Henrich97f].

[6] The data of these benchmark problems can be downloaded from the Web page at http://wwwipr.ira.uka.de/~paro/gkatz/benchmark.html

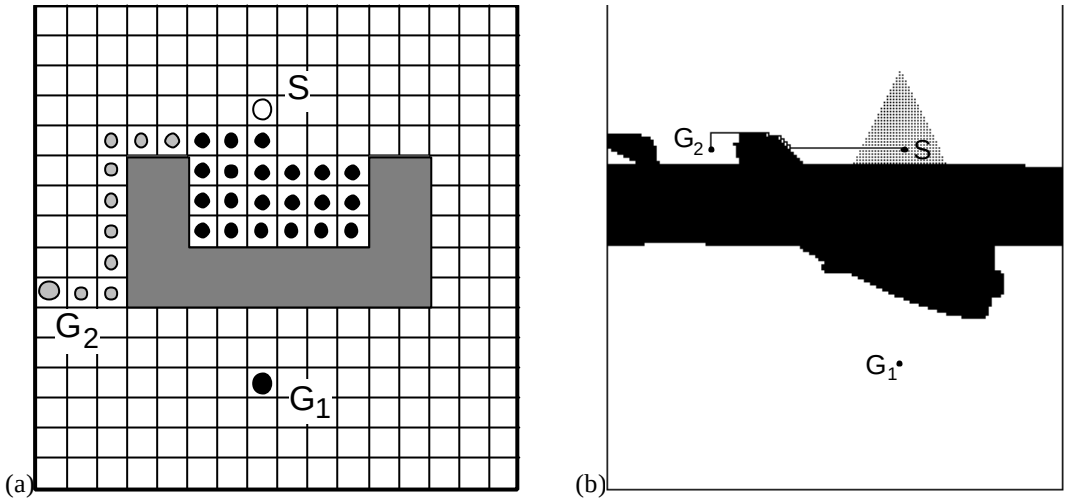


Figure 3: Concept (a) and benchmark SIMPLE (b) as an example for a search in C-space starting at configuration *S* with dynamic switching between multiple goal configurations $G_1$ and $G_2$. The shading of the dots in (a) indicates the goal currently selected.

## 3 Goal switching

Previous run-time measurements show that the planning times are quite different for the different goal configuration. They fluctuate between fractions of seconds up to the insolubility of the task. However, it is impossible for the user to recognize beforehand which goal is favourable and which not. Thus, a method is required to cope with this problem during planning time. In the following, three different possibilities are shortly discussed. For details, see [Beeh97].

The simplest approach is to statically select the goal before searching. This is similar to the case where a goal configuration is selected by the user. For selecting a favourable goal, the planning system requires meaningful criteria. In our case, the geometry of the obstacles transformed into the configuration space is unknown, and solely the joint angles of the start and goal configuration are available. Thus, as selection criterion only the joint angle distance between the start configuration and the goal configurations is applicable. This is a rather inappropriate criterion because obstacles frequently bar the direct way and the search must find a long detour. The length of the detour stands in no relationship to the joint angle distance of start and goal. Therefore, this approach is dropped.

Another approach is to introduce an extra search front for each goal. If there are *n* different goal configurations, then *n* paths are searched simultaneously from the start to all goal configurations. On the one hand, the unreachable goals are avoided in the sense that a favourable goal will be found quickly. On the other hand, the planning takes *n* times longer as the standard search for the most favourable goal. This may be acceptable for nearby and a low number of goal configurations, but as the single searches become more time consuming, the overall planning time will take too long. Additionally, the multiple search fronts cannot use a common OPEN-list, since they rate the single nodes differently. Thus, an increasing number of goal configurations and, with this, many OPEN-lists will finally lead to memory overflow.

In the third approach, called *goal switching*, a single search is accomplished as in the original algorithm. If the planning system detects that another goal is more favourable

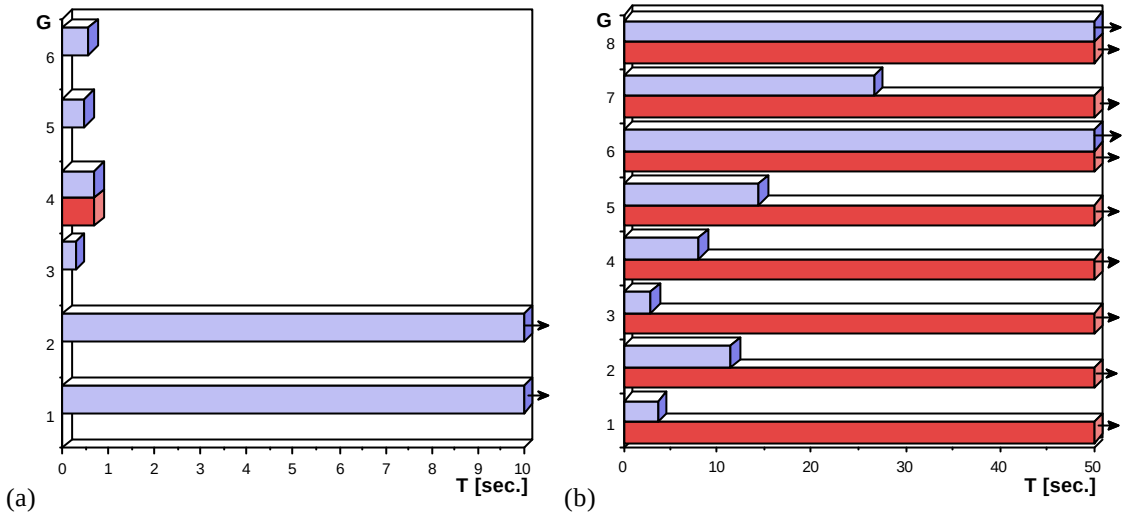


Figure 4: Run-times *T* for search towards fixed goal *G* (light grey) and automatic switching between goals *G* (dark grey) for benchmark problems SIMPLE (a) and STAR (b). The problems, which could not be solved due to memory overflow, are indicated by an arrow.

while searching, it will switch its search direction to the new goal (see Figure 3a). Thus, the search always selects the currently best goal. This switching can be realized very simply by a small modification in the heuristic evaluation *h(n)* of 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 configuration 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 towards 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 an other goal is expanded. The prior goal is dropped and the search switches to the new goal. Figure 3b illustrates how the search towards the first goal $G_1$ results in a surrounding operation until the search front approaches goal $G_2$ so that it is now more favorable.

For the experimental results, the efficiency of goal switching depends greatly on the computation of the heuristic, since only it determines which goal is the next most favourable. We have to choose some reasonable parameter setting because the best setting is not known beforehand. Thus, we choose the Airplane distance as *h()* and weight $w = 0.99$ between *g()* and *h()*.

The experimental results for the goal switching approach are shown in Figure 4. For benchmark SIMPLE in Figure 4a, the planning method selects the fourth goal and finds a path in approximately the same time as the basic method. Thus, the additional calculations of the goal switching cause an only slightly additional expenditure. For benchmark STAR in Figure 4b, the planning method tries to find a path to a goal that is difficult to reach and fails thereby. Thus, the goal switching approach has to cope with the problem, that the selection of the most favourable goal is based on a rather vague heuristic. An inappropriate selection will finally be corrected during the search. However, this may already be too late. It may happen that the searching towards an unfavourable goal wastes too much effort, and the planning system fails due to memory overflow.

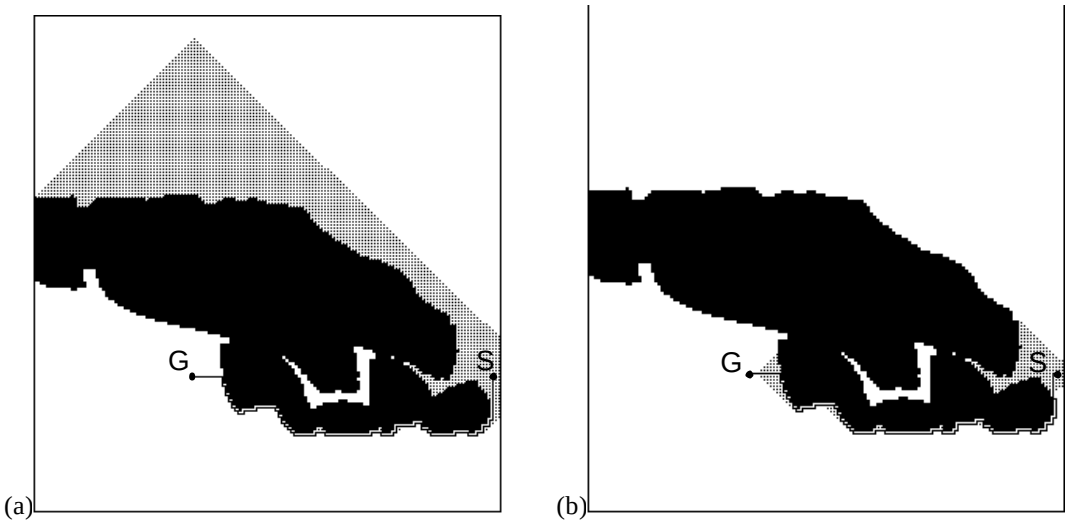


Figure 5: Example for uni-directional (a) and bi-directional (b) search in C-space between start configuration *S* and goal configuration *G*. (2D slice of benchmark DETOUR from [Katz96])

# 4 Bi-directional 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 *bi-directional search* performs both search directions simultaneously. For example, see Figures 5 and 9. The search task is finished as soon as the two search fronts meet each other. Then, the path of the forward search is connected to the inverse path of the backward search.

There are two main advantages offered by the bi-directional search as compared to the uni-directional version. First, depending on the given problem, the backward search can be much simpler than the forward search usually performed. This is the case, for example, in Figure 5. Second, there can be improvements in the run-time complexity. A uni-directional breadth-first search in a graph with branching factor $b$ and length $l$ of the solution path has a complexity of $O(b^l)$. The bi-directional search has a complexity of $O(b^{l/2})$ if both search fronts meet approximately in the middle of the path [Nelson92]. This corresponds to an exponential run-time reduction.

Unfortunately, this reduction does not fully hold for the best-first search, because it operates in a more goal-oriented way than the breadth-first search due to the heuristic. Thus, the search tree with depth $l$ is not completely explored. This effect is reinforced by the necessary weight $w$ of the heuristic $h(n)$, which results in an even more goal-directed search (see Section 2). Therefore, the mentioned run-time improvement is less. Additionally, it may happen that the forward and backward searches pass by each other and meet after the middle. This so-called *search anomaly* worsens the run-time of the algorithm up to twice the run-time for a uni-directional search.[7]

For implementing the bi-directional search, there are basically two ways: using one or two OPEN-lists (similar to goal switching in Section 3). In the first way, the nodes of the forward as well as of the backward search are stored in a common OPEN-list. In each

[7] A way out of search anomalies and the resulting problems are outlined in Section 6.

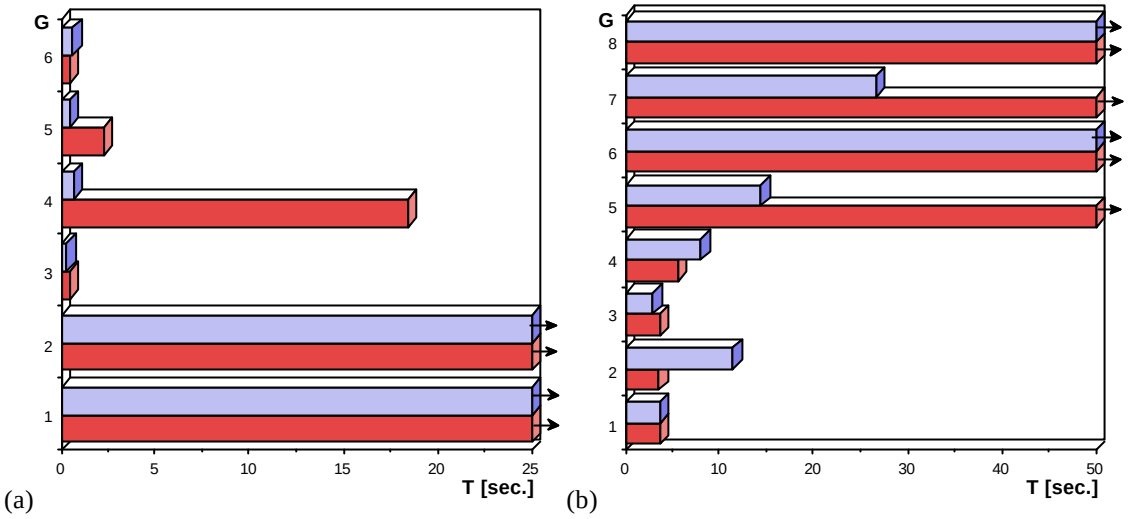


Figure 6: Run-times *T* using the forward search to goal *G* (light grey) and using the backward search from goal *G* (dark grey) for benchmark problems SIMPLE (a) and STAR (b)

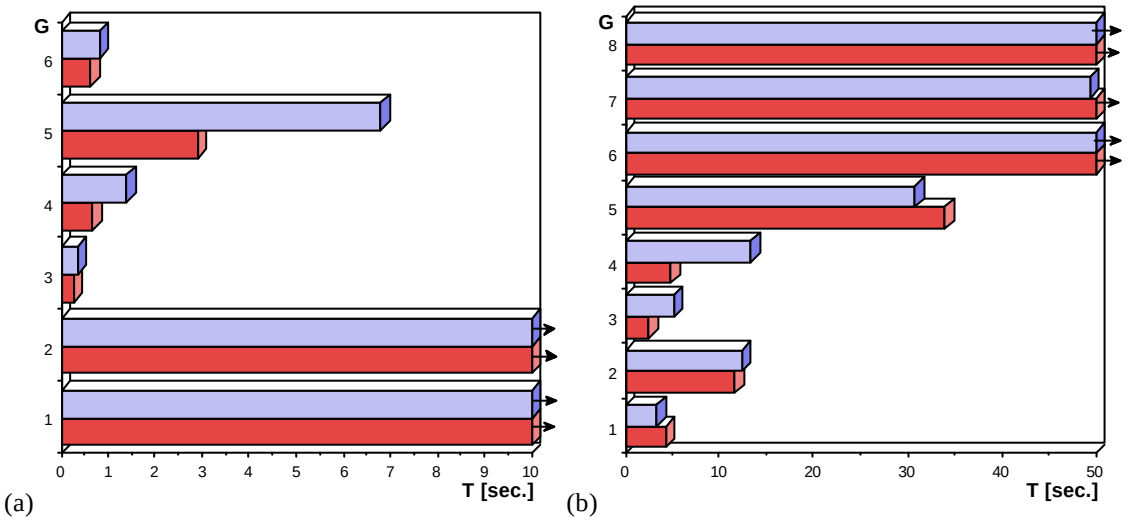


Figure 7: Run-time *T* for bi-directional search with one (dark grey) and two (light grey) OPEN-lists for different goals *G* of benchmark problems SIMPLE (a) and STAR (b)

search iteration, the currently best node is selected from this list independently of search direction it belongs to. This has the advantage that there is only 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 currently 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, the two search fronts will unlikely meet in the middle, thus, the run-time improvement of the bi-directional search gets lost. For the parallel version of the bi-directional 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. For details, see [Beeh97].

In the second way, two separate OPEN-lists are used for the forward and the backward search (confer [Nassimi95]). In each search iteration, the currently best node is selected alternatively from the two lists, thereby processing the forward and backward

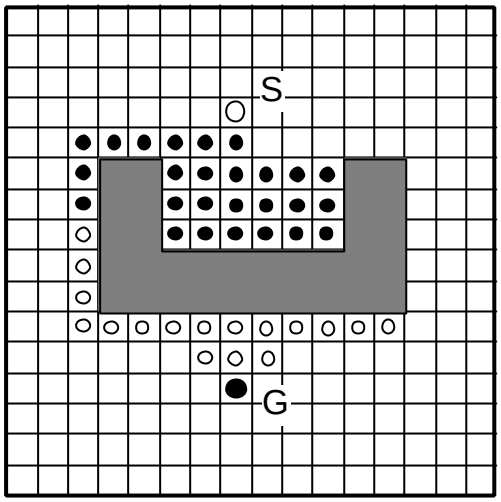


Figure 9: Concept of bi-directional search: One front searches from start *S* to goal *G*; a second front searches from goal *G* to start *S*. The shading of the dots indicate the goal currently selected.

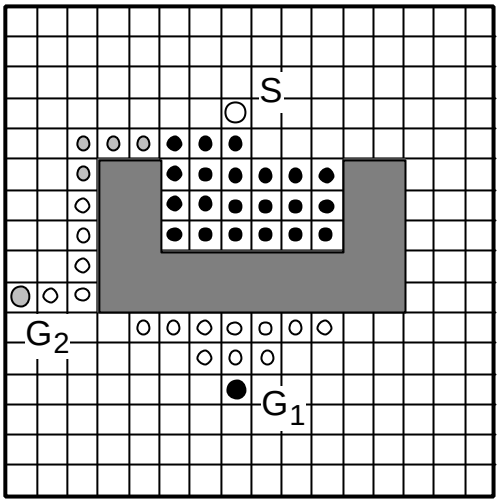


Figure 10: Concept of multi-directional search with goal switching: One front searches forward from start *S* to multiple goals ($G_1$ and $G_2$) with goal switching; Additional fronts searches backward from each goal to the start.

search simultaneously. The overall run-time is at most twice the run-time of the faster version of the forward and backward search processed separately. If the search fronts meet each other before finishing their task, this run-time is sped up.

Experimental results with run-times of the separate forward and the backward search for two benchmark problems are given in Figure 6. The forward direction of benchmark SIMPLE can usually be solved faster than the backward direction. For benchmark STAR, it is mixed. For some search directions of both benchmark problems, no solution could be found by uni-directional search due to memory overflow (indicated by arrows in the figures). These results form the basis for the following comparisons concerning bi-directional search.

Experimental results for the bi-directional search with one and two OPEN-lists are shown in Figure 7. For almost all goal configurations of both benchmark problems, the use of only one OPEN-list is faster than of two lists. The search was able to select the favourable search direction. As an exception, one OPEN-list for Goal 7 of benchmark STAR fails because the planning method pushes the unfavourable direction. Here, two OPEN-lists are successful by simultaneously processing both directions. Comparing to the uni-directional search in Figure 6, the bi-directional search could solve one additional problem. The expected run-time reductions caused by meeting both search fronts could not be validated, which is certainly vested in the high weight *w* of the heuristic *h()*.

## 5 Multi-directional search with goal switching

The approaches presented in the preceding two sections can be combined quite easily. There is one simultaneous forward search from the start to all goals by goal switching. Additionally, there are multiple backward search fronts from all goals to the start, resulting in a *multi-directional search*. For illustration of this concept, see Figure 10. The multiple backward searches use only one common OPEN-list, which has two effects. First, the memory requirement is much less than separate OPEN-lists for each backward search. Second, the currently best appearing search is always pushed ahead at any one time. In the

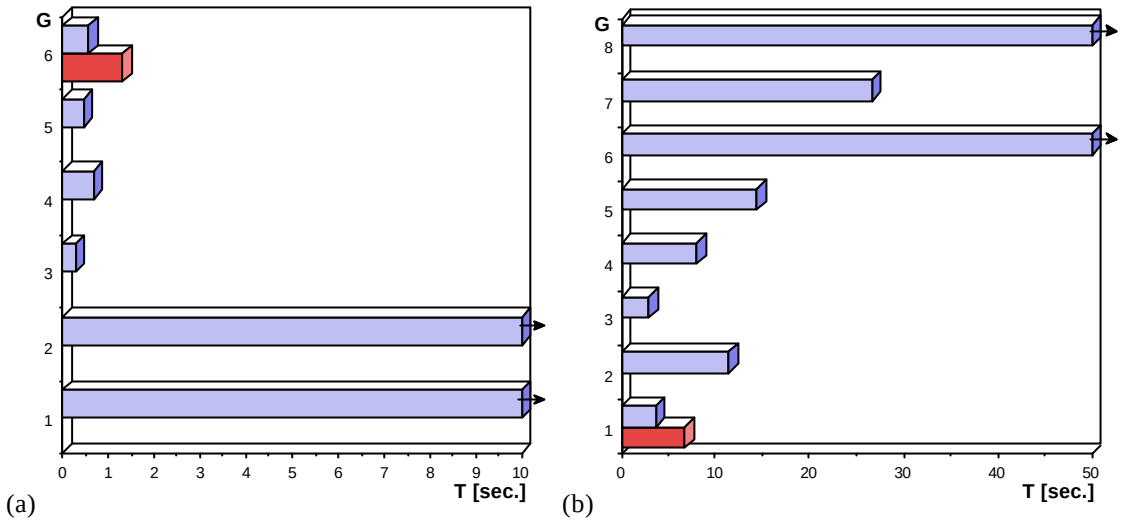


Figure 11: Run-times *T* for uni-directional search (light grey) and multi-directional search with goal switching between goals *G* (dark grey) for benchmark problems SIMPLE (a) and STAR (b)

parallel version, the backward search fronts may be processed simultaneously if the corresponding goal configurations have been mapped on different processors. For details, see [Beeh97].

The experimental results for the multi-directional search with goal switching are shown in Figure 11. For benchmark SIMPLE, the goal configuration 6 is finally selected. It requires a somewhat longer planning time than for goal switching without multi-directional search. This is due to the expansion of alternating nodes of the two OPEN-lists. Still, the run-time is very good, since the unsolvable Goals 1 and 2 are avoided automatically. For benchmark STAR, Goal 1 is finally selected and the unsolvable Goals 6 and 8 are avoided automatically.

## 6 Conclusion and future work

We have presented a parallel path planning system, which is able to automatically handle multiple goal configurations as input. There are two basic approaches, goal switching and bi-directional search, which were combined in the end. Goal switching is done by dynamically switching from one goal to another depending on some distance function. Bi-directional search introduces the probably faster backward search from the goal to the start configuration. The multi-directional search combines these two basic approaches by applying one forward search with goal switching and multiple backward searches, one for each goal.

On the one hand, both basic approaches, goal switching and bi-directional search, have their disadvantages. Goal switching has to cope with a weak heuristic, since sometimes an unfavourable goal is preferred. The bi-directional search has the problem of a double search effort, which is worth it if the backwards search is simpler. On the other hand, the bi-directional search can decrease the effects of the weak heuristic used for goal switching. Goal switching helps bi-directional search by choosing the most simple (nearest) goal. Thus, the multi-directional search integrates the advantages of of the basic approaches. With this, it omits unsolvable goals and achieves short run-time results.

Additionally, multi-directional search enables the planning system to automatically select a favourable goal and to select the shorter search direction by bi-directional search

fronts. Thus, it is no longer necessary that the user selects one of the goal configurations or a search direction. The selected goal is not necessarily the best one, but it certainly is a better choice than the one of the user, since it is based the on-line available information not available for the user.

The presented path planning method can be extended in two ways. First, in the parallel bi-directional search, a wave-shaping mechanism may be useful to avoid possible double search effort. With *wave-shaping*, the search directions of the two search fronts always point to each other. This extension can be realized quite simply for sequential processing. For parallel processing with domain decomposition as a load distribution, it is expected that wave-shaping will cause a high communication effort. Second, in the parallel multi-directional search, an improved load distribution may be useful to avoid idle times at the beginning of the search. This can be done by mapping the hypercubes onto processors such that the start and each goal configuration are mapped onto different processors. In this case, each processor provided with a goal configuration will have useful work from the very beginning.

## References


[Beeh97] Beeh F.: "Erweiterung eines parallelen Bewegungsplaners für dynamische Umgebungen". Master's Thesis, University of Karlsruhe, Faculty for Informatics, 1997.

[Hart68] Hart P. E., Nilsson N. J., Raphael B.: "A formal basis for the heuristic determination of minimum cost paths". In: IEEE Trans. Systems, Science and Cybernetics, Jan. 1968, pp 100-107.

[Henrich92] Henrich D., Cheng X.: "Fast distance computation for on-line collision detection with multi-arm robots", In: Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, May 10.-15., 1992, pp. 2514-2519.

[Henrich97e] Henrich D., Gontermann S., Wörn H.: "Kollisionserkennung durch parallele Abstandsberechnung". In: 13. Fachgespräch Autonome Mobile Systeme (AMS'97), Stuttgart, 6.-7. Oktober 1997, Springer-Verlag, Reihe "Informatik Aktuell".

[Henrich97f] Henrich D.: "Fast motion planning by parallel processing – A review", In: Journal of Intelligent and Robotic Systems, vol 20, no 1, pp 45-69, Sept 1997.

[Hwang92] Hwang Y. K., Ahuja N.: "Gross path planning – A survey", ACM Computing Surveys, vol 24, no 3, Sept. 1992.

[Kamal96] Kamal L., Gupta K., del Pobil A.P.: "Practical path planning in robotics: Current approaches and future directions", IEEE Robotics & Automation Magazine, Dec. 1996.

[Katz96] Katz G.: "Integration eines parallelen Bewegungsplaners in ROBCAD", Master's Thesis, University of Karlsruhe, Faculty for Informatics, 1996.

[Korf92] Korf R. E.: "Search". In: Encyclopaedia of Artificial Intelligence, S. C. Shapiro (ed.), vol 2, pp 1460-73, 2nd Edition, John Wiley&Sons, New York, 1992.

[Nassimi95] Nassimi D., Joshi M., Sohn A.: "H-PBS: A hash-based scalable technique for parallel bidirectional search". In: Proc. of the 7th IEEE Symposium on PDP, San Antonio, 1995, pp 414-421.

[Nelson92] Nelson P. C., Toptsis A. A.: "Unidirectional and bidirectional search algorithms". In: IEEE Software, vol 9, no 2, 1992, pp 77-83.

[Wurll97a] Wurll C., Henrich D.: "Ein Workstation-Cluster für paralleles Rechnen in Robotik-Anwendungen". In: Proceedings der 4. ITG/GI-Fachtagung Arbeitsplatz-Rechensysteme (APS'97), Universität Koblenz-Landau, 21.-22. Mai 1997, pp 187-196.

[Wurll98a] Wurll C., Henrich D., Wörn H.: "Parallel on-line path planning for industrial robots". In: Proceedings of Robotics 1998: The Third ASCE Specialty Conference on Robotics for Challenging Environments, Albuquerque, New Mexico, April 26.-30., 1998.