Open access • Proceedings Article • DOI:10.1109/IROS.1993.583784

The "Ariadne's clew" algorithm: global planning with local methods

— Source link

Pierre Bessière, Juan-Manuel Ahuactzin, El-Ghazali Talbi, Emmanuel Mazer

Published on: 26 Jul 1993 - Intelligent Robots and Systems

Topics: Search algorithm, Motion planning, Path (graph theory) and Genetic algorithm

THE "ARIADNE'S CLEW"[1] ALGORITHM:

GLOBAL PLANNING WITH LOCAL METHODS[2]

Pierre BESSIÈRE[3], Juan-Manuel AHUACTZIN, El-Ghazali TALBI & Emmanuel MAZER

CNRS[4]
IMAG Institute, LIFIA[5] & LGI[6] laboratories

ABSTRACT

The goal of the work described in this paper is to build a path planner able to drive a robot in a dynamic environment where the obstacles are moving.

In order to do so, we propose a method, called "ARIADNE'S CLEW algorithm", to build a global path planner based on the combination of two local planning algorithms : an EXPLORE algorithm and a SEARCH algorithm. The purpose of the EXPLORE algorithm is to collect information about the environment with an increasingly fine resolution by placing landmarks in the searched space. The goal of the SEARCH algorithm is to opportunistically check if the target can be easily reached from any given placed landmark.

The ARIADNE'S CLEW algorithm is shown to be very fast in most cases allowing planning in dynamic environments. Hence, it is shown to be complete, which means that it is sure to find a path when one exists. Finally, we describe a massively parallel implementation of this algorithm.

INTRODUCTION

The goal of this work is to build a path planner able to drive a robot in a dynamic environment where the obstacles are moving.

Designing a path planner is a central question in robotics research. A review of the existing approaches can be found in Latombe's book [1]. There are two main ways to deal with this problem : the global and the local approaches. The global approaches suppose that a complete representation of the configuration space has been computed before looking for a path. The global approaches are complete in the sense that if a path exists it will be found. Unfortunately, computing the complete configuration space is very time consuming, worst, the complexity of this task grows exponentially as the number of degrees of freedom increases. Consequently, today most of the robot path planners are used off-line : the planner is invoked with a model of the environment, it produces a plan which is passed to the robot controller which, in turn, executes it. In general, the time necessary to achieve this, is not short enough to allow the robot to move in a dynamic environment. The local approaches need only partial knowledge of the configuration space. The decisions to move the robot are taken using local criteria and heuristics to choose the most promising directions. Consequently, the local methods are much faster. Unfortunately, they are not complete, it may happen that a solution exists and is not found. The local approaches consider planning as an optimisation problem, where finding a path to the target corresponds to the optimisation of some given function. As any optimisation technique, the local approaches are subject to get trapped in some local optima, where a path to the goal has not been found and from which it is impossible or, at least, very difficult to escape.

The ultimate goal of a planner is to find a path in the configuration space from the initial position to the target. However, while searching for this path, an interesting sub-goal to consider may be to try to collect information about the free space and about the possible paths to go about that space. The ARIADNE'S CLEW algorithm tries to do both at the same time. An EXPLORE algorithm collects information about the free space with an increasingly fine resolution, while, in parallel, a SEARCH algorithm opportunistically checks if the target can be reached. The EXPLORE algorithm works by placing landmarks in the searched space in such a way that a path from the initial position to any landmark is known. In order to learn as much as possible about the free space the EXPLORE algorithm tries to spread the landmarks all over the space. To do so, it tries to put the landmarks as far as possible from one another. For each new landmark produced by the EXPLORE algorithm the SEARCH algorithm checks with a local method if the target may be reached from that landmark. The ARIADNE'S CLEW algorithm is very fast, however, we will show that it is a complete planner which will find a path if one exits. The resolution at which the space is scanned and the time spent to do so, automatically adapts to the difficulty of the problem.

Both the EXPLORE and the SEARCH algorithms may be seen as solving optimisation problems. We first introduce the optimisation technique we are using, namely, genetic algorithms. We then describe successively in some details, the SEARCH algorithm, the EXPLORE algorithm and the concatenation of both. We finally explain a massively parallel implementation of our method and present some results proving that using this method we are able to drive a robot in a dynamic environment. We conclude with a discussion and some perspectives for future work.

PRINCIPLE OF GENETIC ALGORITHMS

Genetic algorithms are programs used to deal with optimisation problems. They have first been introduced by Holland [2]. Their goal is to find optimum of a given function F on a given search space S. For instance, the search space S may be 2^N, a point of S is then described by a vector of N bits and F is a function able to compute a real value for each of the 2^N vectors.

In an initialisation step a set of points of the search space S (called a "population" of "individuals"), is drawn at random (the "genotype" of each individual is a vector of N bits). Then, the genetic algorithm iterates over the following 4 steps until a satisfying optimum is reached (see figure 1 below) :

1 - **Evaluation**: The function F is computed for each individual, ordering the population from the worst to the best.

2 - **Selection**: Pairs of individuals are selected, best individuals having more chance to be selected than poor ones (one individual may appear in different pairs).

3 - **Reproduction**: New individuals (called "offspring") are produced from these pairs.

4 - **Replacement**: A new population is generated by replacing some of the individuals of the old population by the new ones.

Reproduction is done using some "genetic operators". Number of them may be used but the two most common are mutation and crossing-over. The mutation operator picks at random some mutation locations among the N possible sites in the vector and flip the value of the bits at these locations as represented in figure 2.

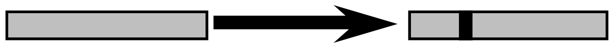


figure 2: mutation operator

The cross-over operator selects at random a cut point among the N possible sites in the binary genotype and exchanges the last parts of the two parents vectors as shown in figure 3.

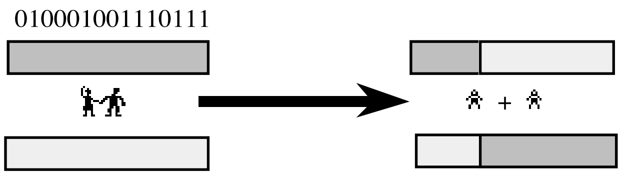


figure 3: cross-over operator

Genetic algorithms have many applications and exhibit very impressive optimisation capabilities compare to other optimisation techniques especially when the search space is big ($\approx 2^{300}$) and F quite irregular (see [3] for a recent survey).

Beside their scientific interest as a model of biological evolution, genetic algorithms have two main technological interests:

1 - They are very robust techniques able to deal with a very large class of optimisation problems.

2 - They are very easy to program in parallel and the acceleration obtained by doing so is considerable (see [4]).

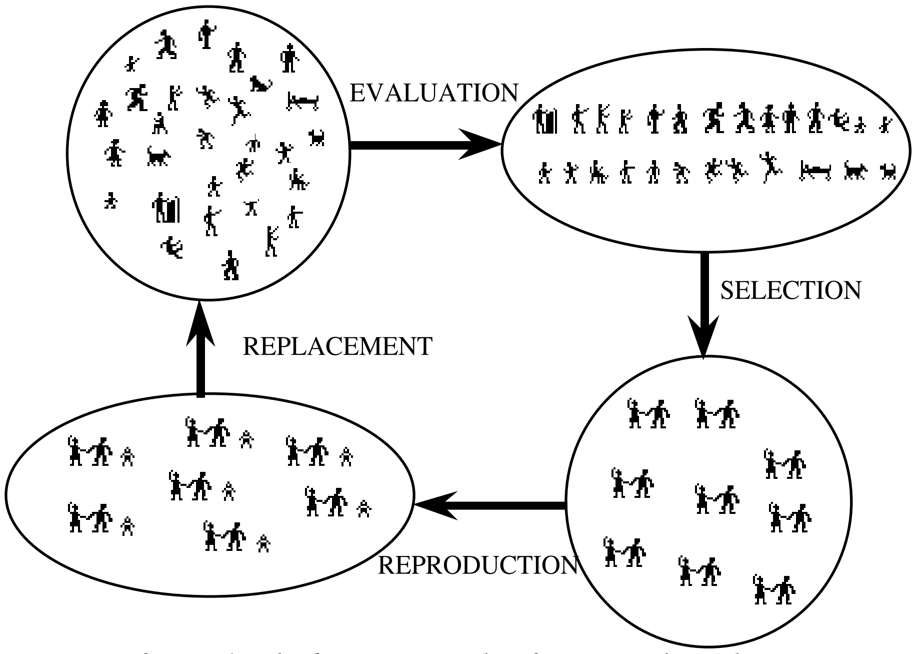


figure 1: The basic principle of genetic algorithms

We proposed a parallel genetic algorithm and developed an implementation on a massively parallel machine based on Transputers (see [5]). This algorithm and the performances obtained by the parallel implementation have been an essential achievement for the success of the work described in this paper.

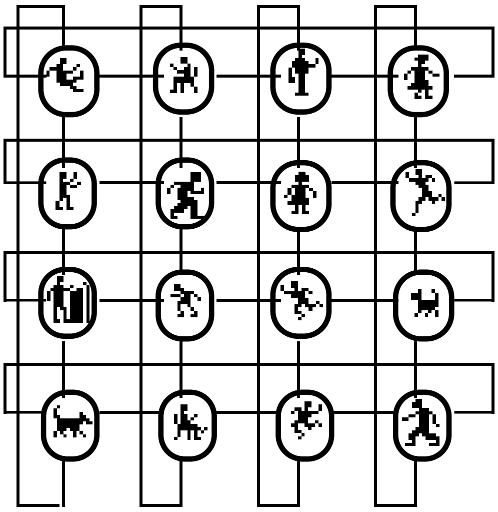


figure 4: the principle of the parallel genetic algorithm

The principle of this parallel genetic algorithm is described by figure 4. It consists in one parallel process running for each individual in the population. The processes are organised in a torus structure where each process has 4 neighbours. At each generation all the individuals, in parallel, choose among their 4 neighbours with whom they want to breed and reproduce with the chosen bride. The parallel genetic algorithm iterates over the following 4 steps until a satisfying optimum is reached :

1 - **Evaluation**: Evaluate *in parallel* all the individuals.
2 - **Selection**: Select *in parallel*, among the four neighbours, the bride with the best evaluation.
3 - **Reproduction**: Reproduce *in parallel* with the chosen bride.
4 - **Replacement**: Replace *in parallel* the parents by the offspring.

THE SEARCH ALGORITHM

The purpose of the SEARCH algorithm is to determine if the target τ may be reached "simply" from a given point π. In order to do so, it looks for fixed length Manhattan motions in the configuration space starting at π and ending at τ.

Given a system with N degrees of freedom $\{\theta_1, \theta_2, ..., \theta_N\}$, a Manhattan motion of length 1 consists in moving each degree of freedom θ_i successively once by $\Delta\theta_i$. A Manhattan motion of length L is a succession of L Manhattan motions of length 1 or of LxN elementary motions of a single degree of freedom. Such a Manhattan motion M is denoted :

$$M = \left(\Delta\theta_1^1, \Delta\theta_2^1, ..., \Delta\theta_i^1, ..., \Delta\theta_N^1, \Delta\theta_1^2, \Delta\theta_2^2, ..., \Delta\theta_N^L\right)$$

Let us call τ_i^j the point reached in the configuration space after ixj elementary motions. Let us call τ_a^b the furthest point reached along M before a collision occurred. We are looking for a collision free Manhattan motion such that $\tau_a^b = \tau_N^L = \tau$.

The SEARCH algorithm may be expressed as an optimisation problem for the parallel genetic algorithm where:

- The search space S_s is the set of all Manhattan motions of length L starting at π.
- The evaluation function F_s applied to a Manhattan motion M given a target τ is defined as follow :

$F_s(M,\tau)=0$ if any τ_i^j of M preceding τ_a^b is in the BACKPROJECTION$_s$ of τ. (The BACKPROJECTION$_s$ of τ is the set of all points of the searched space from which τ may be reached by a Manhattan motion of length 1).

Otherwise, $F_s(M,\tau)=\|\tau - \tau_a^b\|$.

The SEARCH algorithm tries to minimise the evaluation function $F_s(M,\tau)$ over the search space S_s.

Manhattan motions have been chosen because for the NxL elementary motions of M, it is possible to compute simply in parallel, both the corresponding τ_i^j and the collision-free test on the path from π to τ_i^j (see [6]). Furthermore, in a 3 dimension physical space, the collision-free test itself consists in three processes running in parallel checking respectively that there is no vertex-to-plan collisions, plan-to-vertex collisions and edge-to-edge collisions. Finally, each of these three processes may be expressed as the parallel evaluation of AxB processes where A is the number of elements in the first set of the test (A is the number of vertices, plans or edges) and where B is the number of elements in the second set (B is the number of plan, vertices or edges).

The SEARCH algorithm may be used as a planner by itself. It has been used as such for several applications. Let us describe briefly two of them (a more detailed presentation may be found in [7]).

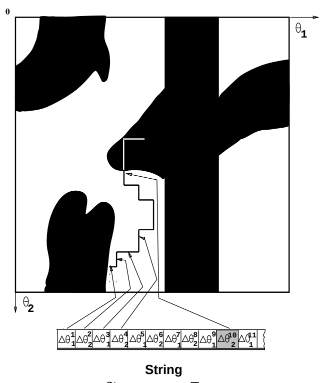


figure 5.a

The first application is a planner for a planar arm with two degrees of freedom. By restricting ourselves to two dimensions we can graphically represent the configuration space and give the reader a better feeling of the method. However, the proposed method does not make any hypothesis about the number of degrees of freedom and can be used without modification for arms with a much larger number of degrees of freedom. Figure 5.a shows a Manhattan motion in the configuration space and the associated "individual" of the genetic algorithm. Figure 5.b shows the initial and final configuration of the arm in the operational space. Figure 5.c shows the path found in the operational space and Figure 5.d the path found in the configuration space. Finally, Figure 5.e shows the portion of the configuration space which has been evaluated. It should be noticed that only a very restricted part of the configuration is really computed, this is one of the main explanation of the efficacy of the algorithm and this is why this algorithm is able to handle planning in dynamic environments.

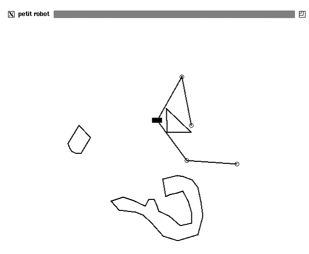


figure 5.b

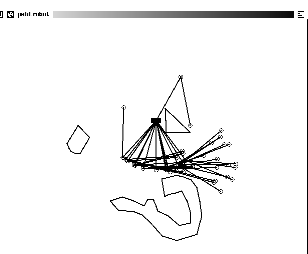


figure 5.c

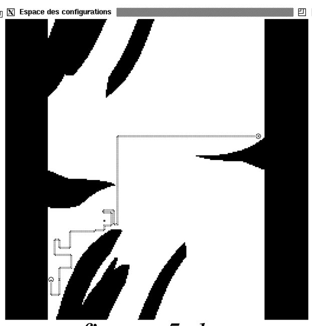


figure 5.d

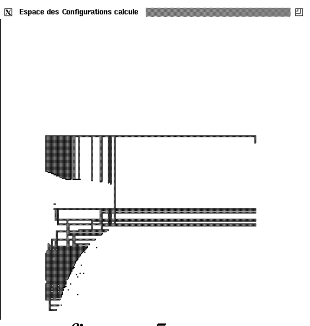


figure 5.e

The second application is a planner for a holonomic mobile robot. Figure 6 shows how the planner behaves in a dynamic environment. Figure 6.a shows the initial found path. Figure 6.b shows the path re-planned after the closing of the door. The

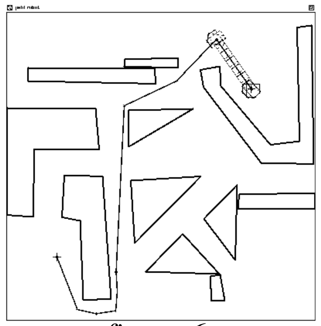


figure 6.a

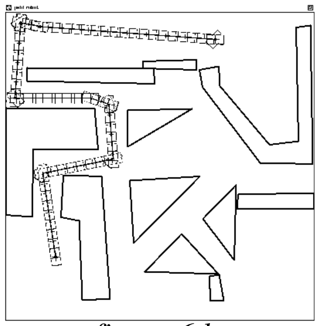


figure 6.b

used version of SEARCH has been implemented on a massively parallel Transputers machine. The planning time for a given path was less than 1 second on a machine of 64 Transputers.

As shown by the two previous examples, the used of SEARCH has a planner is very interesting. However, it may happen that the genetic algorithm gets trapped in some local minima. In that case the planner does not find a solution even if one exists. The SEARCH algorithm is not complete, this is its main drawback.

THE EXPLORE ALGORITHM

The purpose of the EXPLORE algorithm is to collect information about the free space. The EXPLORE algorithm works by placing landmarks in the searched space in such a way that a collision free Manhattan path from the initial position π to any landmark λ_k is known. In order to learn as much as possible about the free space the EXPLORE algorithm tries to spread the landmarks all over the space. To do so, it tries to put the landmarks as far as possible from one another. Let us call $\Lambda = \{\lambda_1,\lambda_2,...\lambda_k,...\}$ the set of already placed landmarks at a given step of the program. It is possible to define the distance between a point α of the searched space and the set Λ by $D(\alpha,\Lambda)=\text{Min} \|\lambda_k - \alpha\|$ on all landmarks $\lambda_k \in \Lambda$.

The EXPLORE algorithm may be expressed as an optimisation problem for the parallel genetic algorithm where:

- The search space S_e is the set of all Manhattan motions of length L starting from any landmark λ_k of Λ.
- The evaluation function F_e applied to a Manhattan motion M of S_e is defined as follows :
  $F_e(M)=D(\tau_a^b,\Lambda)$ where τ_a^b is still the furthest point reached along M before a collision occurred.

The EXPLORE algorithm tries to maximise over the search space S_e the evaluation function $F_e(M)$.

Figure 7 shows how the landmarks spread in the environment.

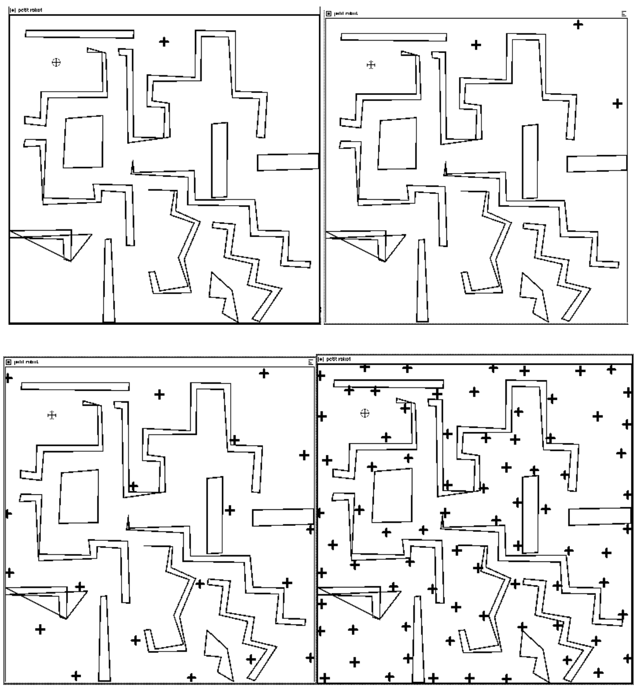


figure 7

Figure 8 shows "the ARIADNE'S CLEW" : a tree of landmarks allowing to go about the free space.

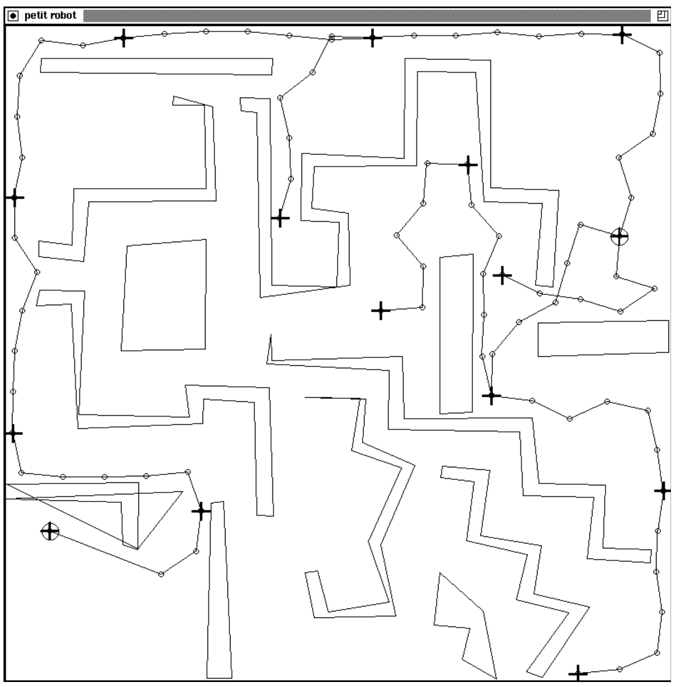


figure 8: The ARIADNE'S CLEW

THE ARIADNE'S CLEW ALGORITHM

The purpose of the ARIADNE'S CLEW algorithm is to find a path from a given point π to a target τ.

The ARIADNE'S CLEW algorithm is the following:

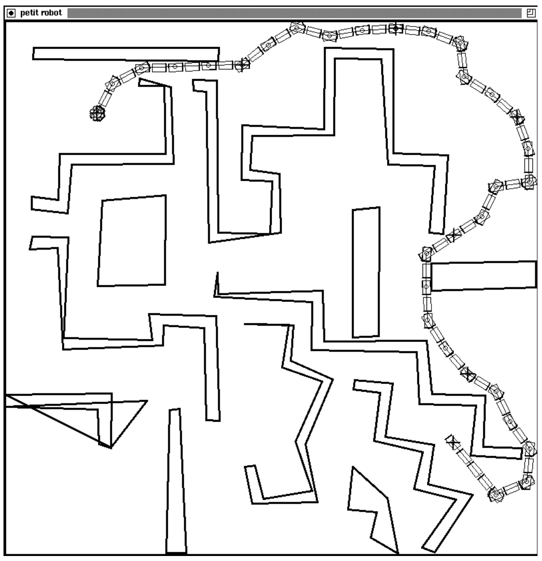
```
1 -  Use the SEARCH algorithm to find if a
     "simple" path exist between $\pi$ and $\tau$.
2 -  If no "simple" path found by step 1, then
     do until a path is found
         2.1  *  Use EXPLORE to generate a new
                 landmark $\lambda$.
         2.2  *  Use SEARCH to look for a "sim
                 ple" path from $\lambda$ to $\tau$.
```

It is interesting to notice that SEARCH may be seen as a backprojection function for EXPLORE. SEARCH could be called BACKPROJECTION_e because it plays relatively to EXPLORE the exact same role than BACKPROJECTION_s relatively to SEARCH. A quite complicated backprojection function indeed, which usually produces very big backprojection allowing EXPLORE to stop after placing just a few landmarks.

The ARIADNE'S CLEW algorithm has three very important qualities:

- It reduces to the very fast SEARCH algorithm for most of the cases.
- It is complete, in the sense that if a path exists it will be found (see proposition 2 below).
- It automatically adapts the resolution at which it scans the space to the complexity of the problem (see proposition 3 below).

Figure 9 shows two complex paths found by the ARIADNE'S CLEW algorithm.



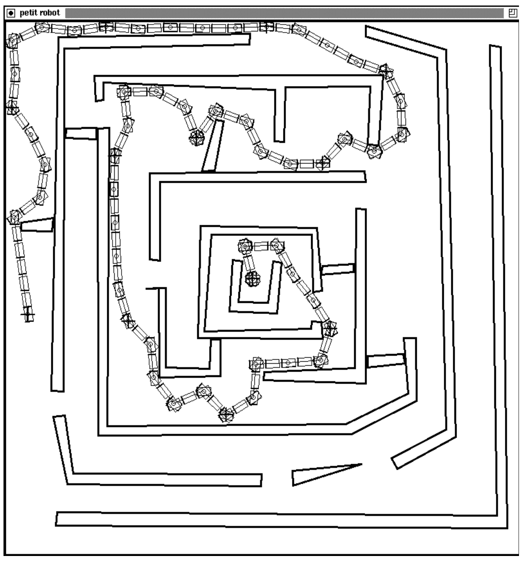


figure 9

In the sequel of this section, three important propositions concerning the ARIADNE'S CLEW algorithm will be established. However, given the restricted length of this paper, only sketches of proofs are proposed.

Definition 1: a PATH P from an initial point π to a target τ in an N dimensions metric space is defined as an N-uplet (F1(t), F2(t), ..., FN(t)) of N continuous functions from [0,1] -> $\Re$ such that (F1(0), F2(0), ..., FN(0)) are the co-ordinates of π and (F1(1), F2(1), ..., FN(1)) are the co-ordinates of τ.

Definition 2: a MANHATTAN MOTION OF LENGTH 1 in an N dimensions space is defined as an N-uplet $\mathcal{M}_1 = \left(\Delta\theta_1^1, \Delta\theta_2^1, \ldots, \Delta\theta_i^1, \ldots, \Delta\theta_N^1\right)$ where each $\Delta\theta_i^1$ is an integer corresponding to the length of the move along dimension i expressed in some given elementary length unit υ.

Definition 3: a MANHATTAN MOTION OF LENGTH L in an N dimension space is defined as an NxL-uplet:
$\mathcal{M}_L = \left(\Delta\theta_1^1, \Delta\theta_2^1, \ldots, \Delta\theta_i^1, \ldots, \Delta\theta_N^1, \Delta\theta_1^2, \Delta\theta_2^2, \ldots, \Delta\theta_N^L\right)$
where each $\Delta\theta_i^j$ is an integer corresponding to the length of the jth move along dimension i expressed in some given elementary length unit υ.

Proposition 1: for any $\varepsilon>0$, for any path P, it is possible to find υ, L and a Manhattan motion M_L of length L, such that the path P is approximated by M_L with an error less than ε.

Sketch of proof:
- Direct application of the Stone-Weirstrass theorem.

Proposition 2: the ARIADNE'S CLEW algorithm is complete, which means that, for any given $\varepsilon>0$, if a path exists from the initial point π to the target τ it will find (in a finite time) L and a Manhattan motion of length L M_L starting at π and ending at τ with an error less than ε.

Sketch of proof:
- Proposition 1 insures that such a Manhattan motion M_L exists.
- The ARIADNE'S CLEW algorithm searches a discrete finite space.
- The ARIADNE'S CLEW algorithm insures that all the produced Manhattan motions are different.

Consequently, M_L will be produced after a finite amount of time.

Remark: In fact Proposition 2 proves that any algorithm producing Manhattan motions without producing twice the same is complete. This is true either for an algorithm enumerating the Manhattan motions or for an algorithm drawing randomly the Manhattan motions (without drawing twice the same). Of course the ARIADNE'S CLEW algorithm is doing much, much, better than those two.

Definition 4: for a given ε, let us call the COMPLEXITY OF THE PROBLEM the minimum number C of identical tiles necessary to do a paving of the space, the biggest dimension of a tile being equal to ε.

Definition 5: let us call RESOLUTION R the number of landmarks generated by the ARIADNE'S CLEW algorithm to find a solution.

Proposition 3: resolution R is always inferior or equal to complexity C.

Sketch of proof:
- as long as R<C, two different landmarks may not be in the same tile given that the ARIADNE'S CLEW algorithm maximises the distance between the landmarks.
- for R=C, there is exactly one landmark in each tile.
- in that case, it exists a Manhattan motion starting at a distance of π less than ε (starting at the landmark in the same tile than π) and ending at a distance of τ less than ε (ending at the landmark in the same tile than τ).

Remark: In practice, experiences prove that R<<C. There are two main reasons for this. First, most of the time, SEARCH stops EXPLORE after the generation of just a few landmarks. Second, the

ARIADNE'S CLEW algorithm adapts locally its resolution to the surrounding free space, generating a lot of landmarks where narrow doors or corridors have to be found and generating just a few of them when in an open free space.

PARALLEL IMPLEMENTATION

It is possible to design a massively parallel implementation of the ARIADNE'S CLEW algorithm with 5 embedded levels of parallelism (see figure 10) :

1 - At top level of parallelism, SEARCH and EXPLORE may run in parallel. EXPLORE produces a landmark while SEARCH exploits the previous one. SEARCH stops EXPLORE as soon as it reaches the target.
2 - Both SEARCH and EXPLORE need to run a genetic algorithm which can be implemented in parallel as described in section 2.
3 - Fs and Fe, the evaluation functions of SEARCH and EXPLORE, consist mainly in testing collision on paths. This may be done by NxL parallel processes, where N is the number of degrees of freedom and L the length of the considered Manhattan motions.
4 - Each of this NxL processes may be further decomposed as three parallel processes testing respectively vertex-to-plan, plan-to-vertex and edge-to-edge collisions.
5 - Finally, each of this test needs AxB parallel processes where A is the number of elements in the first set of the test (A is the number of vertices, plans or edges) and where B is the number of elements in the second set (B is the number of plans, vertices or edges).

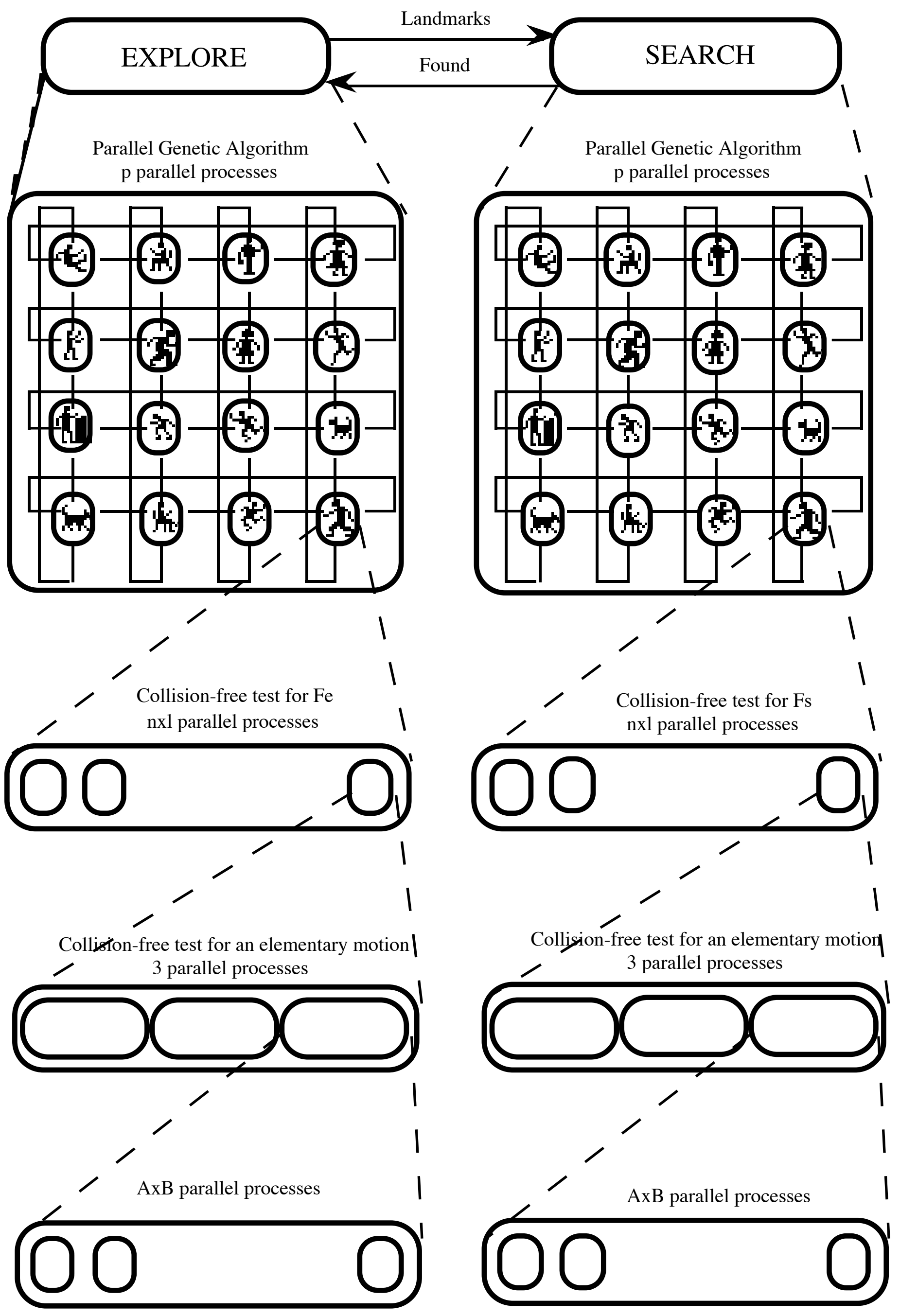


figure 10

The methodology used to implement this on a parallel machine consists in writing the application fully in parallel as if there were as many processors available as the number of processes. Of course, in practice, this is not the case. However, we use languages and tools (particularly the PAROS parallel operating system and the PARX communication kernel developed in the SUPERNODE II project, see [8]) which allows to conceive a parallel program independently of the architecture of the target machine.

We implement this on SUPER-NODE machine made of 128 Transputers.

CONCLUSION, RESULTS AND PERSPECTIVES

We have presented a general method to search a continuous configuration space. This method is implemented using a minimisation technique based on parallel genetic algorithms. We have demonstrated the validity of the method on a set of complex path planning problems. Finally, we proposed a massively parallel implementation of the method which permits on-line re-planning.

Our experimental set-up used to test our algorithm for an actual six degrees of freedom arm is represented on figure 11. RobotI is under the control of the MegaNode (128 T800 Transputers parallel machine) running a parallel implementation of the ARIADNE'S CLEW algorithm. RobotII is used as a dynamically obstacle : it is manually controlled via KALI (KALI is a robot control software initially developed at Mc Gill University). First we use a CAD system called ACT[7] which permits precise geometrical description of the arms and obstacles and which is able to present 3D simulations. We compile this representation into a special format which is downloaded to the MegaNode. A final position is then specified to RobotI, the MegaNode quickly (**2 seconds**) produces a plan which assume RobotII is standing still, should the position of RobotII change under manual control, RobotI stops and the MegaNode (re)computes another path.

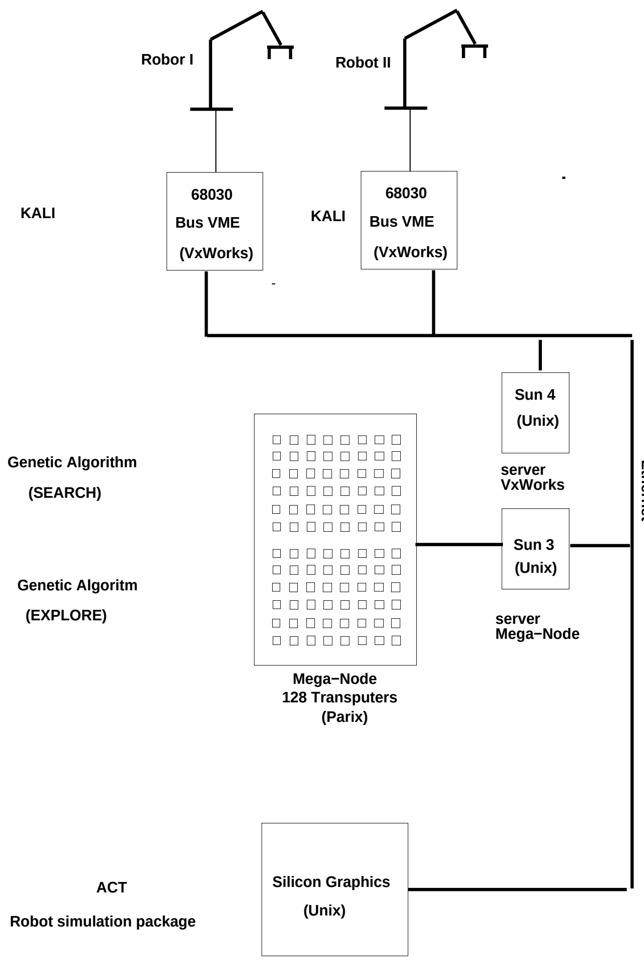


figure 11

We plan in the next months to work on the problems of grasping, re-grasping and fine motion synthesis.

BIBLIOGRAPHY

[1] J-C. Latombe; "*Robot motion planning*", Ed. Kluwer Academic Publisher, 1991.

[2] J.H. Holland; "*Adaptation in natural and artificial systems*"; Ann Arbor: Univ. of Michigan Press, 1975.

[3] D.E. Goldberg; "*Genetic algorithms in search, optimization, and machine learning*"; Addison-Wesley, 1989.

[4] E-G. Talbi & P.Bessière; "*A parallel genetic algorithm for the graph partitioning problem*"; ACM Int. Conf. on Supercomputing, Cologne, Germany, June 1991.

[5] E-G. Talbi & T.Muntean; "*A parallel genetic algorithm for process-processors mapping*", Int. Conf. on High Speed Computing II, Montpellier, M.Durand and F.El Dabaghi (Editors), Elsevier Science Pub., North-Holland, pp.71-82, Oct 1991.

[6] T. Lozano-Pérez, J.L. Jones, E. Mazer & P.A. O'Donnell; "*HANDEY A robot task planner*"; the MIT Press, 1992

[7] J.M. Ahuactzin, A-G. Talbi, P. Bessière & E. Mazer; "*Using genetic algorithms for robot motion planning*"; ECAI92, Vienna, Austria, 1992

[8] T. Muntean, N. Gonzalez & Y. Langue; "*PARX Kernel for the PAROS parallel operating system*"; ESPRIT '91; Kluewer Academic Publishers, 1991.

[1]Once upon a time, Ariadne, daughter of Minos, King of Crete, helped Theseus to kill the Minotaur who lived in the Labyrinth, a huge maze build by Daedalus. The main difficulty for Theseus was to find his way through out the Labyrinth. Ariadne imagined to give him a thread to unwind in order to find his path back.

[2]This work has been made possible by two EEC ESPRIT project: SUPERNODE II (n°2528) & PAPAGENA (n°6857) and CONACYT Mexico

[3]Telephone: (33)76.57.46.73; E-mail: bessiere@imag.fr; Fax: (33)76.57.46.02.

[4]Centre National de la Recherche Scientifique

[5]Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle

[6]Laboratoire de Génie Informatique

[7]Kali andACT are comercial products of Aleph Technologies

Paper published in **IEEE-IROS'93** conference
Intelligent **RO**bots and **S**ystems
Yokohama - JAPAN - 1993