

1. An overview and a purpose of the monograph

This monograph concerns one of the most important problems in ground mobile robotics' autonomy, the Global Path Planning (GPP) – the work does not concern planning for aerial or underwater vehicles. Robot's autonomy of which a GPP is a building block refer to a system capable of operating in the real-world environment without any form of external supervision including control for extended periods of time. This operation includes an interaction of the mobile robot with a surrounding environment, which is inseparable from motion. The essence of the GPP may be summarized as follows: it is a problem of finding a path represented by an ordered set of points belonging to some map, which mobile robot has to consecutively traverse one after another, in order to reach a destination position. Between these intermediary points the execution of movement planned by Local Movement Planning (LMP) may be performed. Not every path that is planned by the GPP algorithm may be valid and the path should be optimized in terms of some criteria as we yield for "best-possible path": the shortest, the most energetically efficient, the quickest etc.

This introductory definition clearly states the inputs of the GPP process to be: a map, a start and a destination points, and the output to be: a set of intermediate points that build up a desired path (trajectory) for the execution of movement. Clearly, not each path through the map is executable because there may be some additional constraints imposed. One seeks for an optimal path, or at least a good one, and hence the GPP problem is of a search and optimization type. The determination of that output, i.e. of an optimal (or, better to say, good) path is the objective of the GPP and the problem considered in this work. Usually, though the mathematical formulation of the GPP problem as an optimization problem is in principle possible, its solution is in virtually all practical cases numerically intractable in view of time requirements, and therefore heuristic methods are frequently used or the problem is somehow simplified.

In the consecutive chapters of this monograph firstly the essence of the GPP problem will be presented, its mathematical problem formulation of an optimization type and difficulties with its solution. Then a review of main solution techniques which are employed for solving GPP is given. The chronological order of appearance have been chosen for presenting approaches, namely starting from computational geometry methods and end up with advanced (meta)heuristics used nowadays. Such approach gives a full frame and present state-of-the-art as well as foreseen trends in GPP field. Since GPP problem flourished in 60' one can expect a plethora of methods for solving this problem. Some of these are very mature and are used in commercial products (such as modifications of A*) algorithm, whilst some are being currently developed with proven promising results on conceptual level. However comparing methods that stem from different principles is difficult, as there are many aspects to take into an account, such a trial have been performed in last chapter of this monograph.

The following work is organized as follows. Chapter 2 introduces Global Path Planning giving basic definitions and presenting critical aspects of search and configuration space. In that chapter a problem is formulated in a mathematical manner and classic map representation techniques are explained. Chapter 3 presents a classical approaches of computational geometry, Artificial Potential Field and graph methods. Chapter 4 presents a current up-to date state-of-the-art of novel heuristics based global Path Planning methods. Chapter 5 is a short view on GPP and presents a summary in a nut shell.

2. Introduction to the Global Path Planning (GPP)

In this introductory chapter, first, a basic definition of the Global Path Planning (GPP) problem is given and then compared with the Local Movement Planning (LMP) problem. Distinguishing between those two is vital in building autonomous ground robot applications. Next, the ideas of a search and configuration spaces and the GPP problem based on those spaces are defined in mathematical terms as a search and optimization problem with constraints. It is important to understand that representation of search and configuration space limits the group of methods that can be used for path planning.

2.1. General problem description

One of the main topics of interest in the field of a ground mobile robot autonomy is strictly connected with the autonomy of motion which is described as the way of how robots plan and execute their movement in order to reach a destination position in a working environment.

The motion autonomy of unmanned ground vehicles, including the mobile robots, falls into two main streams [1]:

- Global Path Planning (GPP), and
- Local Movement Planning (LMP).

It is essential to understand the main difference between them: the Global Path Planning uses infrequently employed and time-intensive methods that are based on a global data and are meant to find a path. The Local Movement Planning, on the other hand, often referred to as the Local Navigation [2] or the Obstacle Avoidance [3], is a computationally fast, often reactive method to proceed within some (often short) specified time and (often restricted) space horizon. These two main categories of problems are key elements for the construction of motion autonomy systems for all types of mobile robots in general.

Though the partitioning of movement autonomy into such two categories can be found inappropriate at the first glance, it has been clearly implied by how humans are moving towards a goal. For instance, when the humans are planning the motion in the mountains, they first look at the ground map to roughly identify possible paths and then going along that path they identify a route using trails. Therefore, the GPP algorithms always have to be paired with the Local Movement Planning (Local Navigator) algorithms. A deeper analysis of both categories will be given later.

The GPP can be defined as a process of using a priori data in the form of a terrain map to find an optimal collision-free way from the start point to a destination point in an environment with obstacles [4]. Another synonymous definition can be found in the literature. For instance, Giesbrecht in [5] argues that the GPP is a process of using accumulated sensor data and a priori information to allow an autonomous robot to find the best path to reach a goal position. Yi-Nan Gao et al. [6] claim that the GPP is defined as finding the most reasonable collision-free route from the start point to the destination point. The key features of each

definition are in line with each other and they consist of a set of basic constituent concepts: a *map*, a *mobile robot*, *start* and *destination* points, *obstacles* and a *collision-free way*. If a path connecting the start and destination points exists, then a path planner should return it, otherwise a failure should be reported.

2.2. Definition of the GPP

The cited definitions [7–9] differ from each other and are not precise but they have one common element: they define the GPP as an optimization problem. To be more precise an exact definition has been formulated as:

The Global Path Planning is defined as a process that – taking as a point of departure some representation of a physical surrounding, called a map, and coordinates of the start and destination points – finds a path represented by an ordered set of points belonging to that map which mobile robot has to consecutively traverse one after another to finally reach a specified destination point, and between those intermediate points the execution of movement planned by the Local Movement Planning methods may be performed. A path of the mobile robot has to be optimized in terms of its length, cost, execution time, etc.

This definition implies the **inputs** of the GPP process to be: a map, start and destination points and the **output**: a set of intermediate points that build up a desired trajectory for the execution of movement. This determination of that output is the objective of the GPP and the **problem considered in this monograph**.

This abstract definition does not consider any particular type of map representation, search space (S-space), configuration space (C-space), robot description or other elements and aspects; they will be defined and elaborated upon further on.

As the path produced by the GPP has to be **optimized**, there is a need for an **evaluation function** that will evaluate the path, and hence to differentiate one path from another considering their quality. One has to notice that not every path through the map that may be determined is executable because there are some additional constraints to be accounted for in that process.

Those elements mentioned: the *objective*, the *representation type*, the *evaluation function*, and the existence of (additional) *constraints* clearly point out that:

The Global Path Planning is a search and optimization problem with constraints.

Therefore: each Global Path Planner is concerned with the following basic issues [10–12]:

- *Configuration space* (C-space). It represents possible states of a mobile robot including coordinates, orientations, etc. In the simplest case the C-space is two-dimensional, based on the Euclidean geometry. Complex planners should also take into account other physical features as, for instance: the state (level) of battery charge, the strength of radio signal, the height above sea level, etc.
- *Search space* (S-space). It represents a set of all candidate solutions to the GPP problem. In general, the GPP problem is NP-complete as in realistic

cases it involves many planning criteria so that the size of the S-space may be considerable.

- *Time*. In simple applications time is not explicitly involved and only a time interval between consecutive points is given. Time is obviously critical for dynamic environments [13] in which the dynamics of environmental changes is comparable with the total time of the path execution.
- *Planning criteria*. The choice of criteria implies the form of the evaluation function for the comparison of different paths. These criteria are optimization criteria but some of them can also constitute constraints. The GPP problem may include: the path length, the battery charge needed for the path execution, the distance to obstacles, execution time, etc. As in each optimization problem, the choice of planning criteria is critical.

2.3. Global Path Planning and Local Movement Planning

In the literature, the following four main tasks of a navigation algorithm are usually listed [14, 15]:

- *Global Path Planning*;
- *Perception*. This means the investigation of surroundings and a proper interpretation of this data. Perception includes the determination of a current mobile robot state;
- *Location*. This means the determination of the current robot position, and for non-point robots it also includes a specific spatial configuration;
- *Local Movement Planning*. This means in general an iterative process of calculating next movement/control signals of the robot due to the calculated global path, current surroundings (in the sense of obstacles to be avoided) and location.

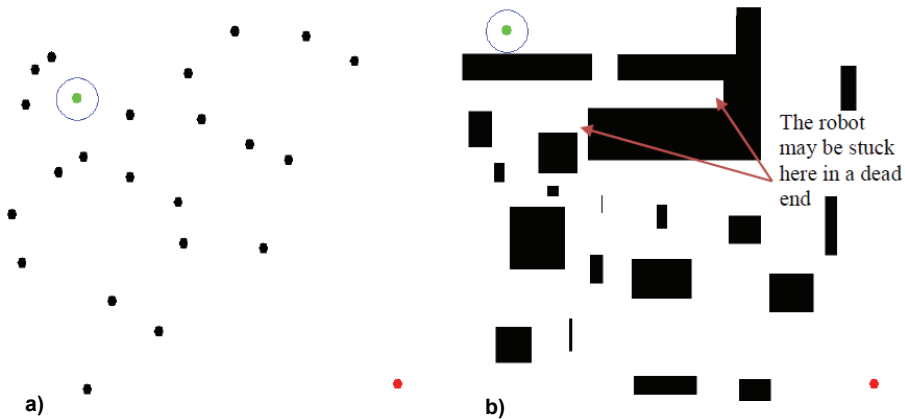


Fig. 1. A mobile robot (the green point) has to reach the destination point (the red point). The blue circle is the border of perception. a) Environment in which the robot can reach the destination point on the basis of the LMP only. b) Environment in which the GPP is essential because of a need for a non-monotonic movement in the sense of a distance to the goal versus the execution time.

The Local Path (Movement) Planning is a continuous process [11] because the control signals have to react to different situations within the surroundings. The local movement execution consist in gathering data from on-board sensors, building an environment representation, locating the robot within that environment, planning of obstacle avoidance behavior, and executing the movement so that the robot follows the required global path [16]. The local obstacle avoidance is often reactive and the more the environment is dynamic the more reactive the algorithm should be. Obviously, the movement should be collision-free, and to be executed in the shortest possible time. In this case the robot knows the environment to an extent that its sensors allow for.

It is important to notice that because of a limited perception ability, there is a strict distinction between the local and global movement planning. Based on sensor signals, the mobile robot can plan its movement only to some extent which is related to the visibility of borders of the surroundings.

A need to use the GPP algorithms depends on the character of the environment. Fig. 1a depicts a map on which the movement could be performed without the GPP. This map does not contain specific artifacts such as blind alleys (dead ends). The latter case (Fig. 1b) presents a map where the GPP is needed due to a limited perception of the robot and a possibility that the robot may be stuck in a dead end¹.

Therefore:

For a general type of environment the use of the GPP algorithms is necessary.

2.4. Search space and configuration space

The search space (S-space) is a set of all possible candidate solutions to a problem considered. The S-space in terms of the GPP identifies the way of how the environment is described, i.e. the S-space defines the location process. Apart from some niche type GPP techniques that transform the search or configuration space using, e.g., the Laplace transformation [17, 18] or hybrid representation [11, 19], the two main types of a map representation are used for the GPP task: topological and metric [20, 21].

The *topological representation* [22] identifies the location of particular objects by a set of specific markers or indicators of the objects that are treated as orientation points. An absolute position of an object does not have to be known as the map contains reciprocal positions only. Reference points and markers create vertices in a graph. Two vertices are connected if a collision-free path exists between them. The GPP problem is in this case considered as a graph-search problem. The location of vertices is derived from information obtained by laser scanners, laser rangefinders, computer vision devices, etc. [23]. Some examples of a topological graph map can be found in [24].

The GPP based on topological maps is executed in two steps: (1) selection of landmark points for graph vertices, and (2) determination of possible edges and

¹ There are some local algorithms that deal with dead ends, but the path produced by them is far from being optimal (see BUG algorithm).

graph search. An optimal path through the graph may not even be optimal as this depends on the landmark selection [25]. A topological representation was developed by Park [25] and Thrun [26].

The *metric representation* [27, 28] consists in assigning to each position of the S-space a set of coordinates. It divides the world into neighbouring cells. The metric representation is environmental-centric that demands proper location methods. A path planned over a metric map is a set of intermediate points the density of which depends on the map granularity. This granularity implies a need for a movement subplanner as in the case of the topological representation. The metric representation uses maps created by employing cell decomposition [29], virtual force fields [30] or graph topology [28].

Some examples of the topological and metrics maps are shown in Fig. 2.

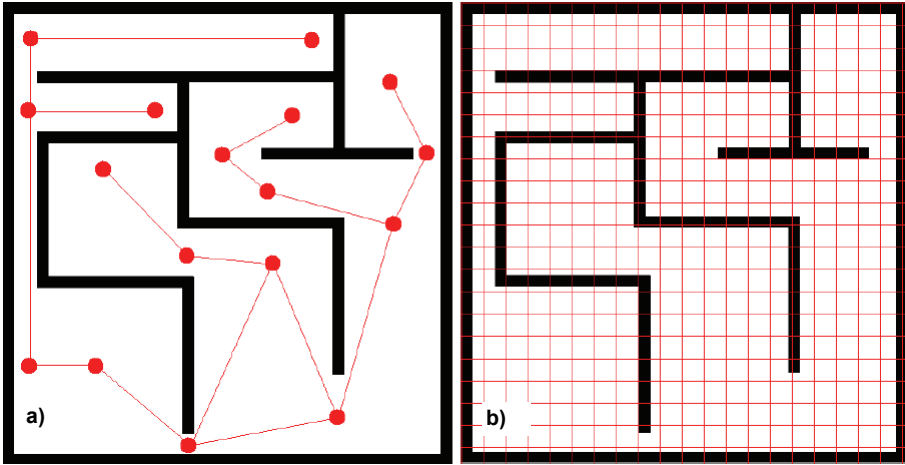


Fig. 2. Example of the S-space for an indoor map: a) a topological map of a landmark type “middle of the room”, “corridor centre”, b) a metric map of an approximate cell decomposition (with a wrong granularity selection).

The *configuration space* (C-space) in the GPP is a set of all possible states of the robot within a particular environment. It may include the robot location, tilt angles, velocity, battery level, etc. In most GPP problems, the C-Space should be simplified to 2 (or 3) dimensions [31]. As Latombe showed in [11], the time complexity of the GPP depends on the number of dimensions of the C-space in an exponential manner. This simplification does not have a negative impact on the GPP because some variables are moved for inclusion into the LMP since in most cases a map provides only a rough estimation of the environment and a lot of features, exemplified by obstacles, are not mirrored there.

2.5. Environment representation methods

A *map* M , which is a representation of the C-space used for the GPP, may represent the surroundings of the robots in one of three main types [32, 33]:

- the *roadmap representation*,
- the *potential field representation*,
- the *cell decomposition*.

In the roadmap representation vertices are points in the C-space and edges connecting these vertices contain well established transitions between the points. The graph in the topological representation deploys vertices in identifiable locations, the *orientation points*. In order to solve the GPP problem one has to find the vertices, define the edges between them and find a permutation of edges leading from the start to the destination vertex. The graph based GPP can be therefore seen as a combination of vertex finding methods which build up the graph, and then solving it.

The benefit from using the roadmap representation approach over the decomposition comes mainly from the number of points for which a planner has to find a path. Its serious drawback is a difficulty in updating a graph needed in the case of a dynamic environment [34] or in the case of using the SLAM (Simultaneous Localization and Mapping Method). To parallelize the algorithm, a proper choice of graph vertices is a critical issue. A lack of valuable vertices leads to an unsolvable graph or to being far from an optimum solution to the GPP problem. All graph methods (see Section 3.3) use some tools to build up lists of vertices such as corners of obstacles or a crossroad of free spaces etc.

The Potential Field Representation method [35–38] is based on the idea, that an environment can be represented by forces acting on a particle (robot) and directing it away from obstacles but towards the goal. Therefore, a robot (a particle) is being pushed away and attracted by the field. For each point of the C-space, the fields forces are to be calculated as a superposition of repel and attract forces. Each field is created according to specific methods and for the description of them we refer the reader to Section 3.2.

In the Cell Decomposition method a map is divided into non-overlapping regions representing the same map feature, i.e. regions of the same characteristics are merged to create one bigger region. Cell decomposition simplifies the environment by reducing the redundancy of information but simultaneously one loses location information as each part of the merged region has one set of coordinates. By merging the regions, a decrease of the number of possible configurations occurs, and this also happens by merging these configurations. Cell decomposition often leads to paths that are far away from optimality even though a method used to generate such a path is optimal. This is because the latter method operates on configurations produced by cell decomposition with itself and therefore loses a location resolution. Cell decomposition itself is not a GPP method but rather a way to simplify the planning by reducing the size of the C-space and therefore the S-space. Cell decomposition usually contributes to graph algorithms by providing the vertices, i.e. through cell identification. Then, the neighbouring function is easy definable and the visibility of a particular cells is clear.

Cell decomposition was first introduced by Brooks and Lozano-Perez [39] and today is the most popular map representation [32]. It divides the environment into non-overlapping areas called cells. Each of these areas represents one value of some feature: the cost of reaching a particular cell, the height above the sea level,

etc. This value reflects how difficult it is to reach that particular cell. The cell decomposition maps are always based on raster images that are already decomposed into pixels. The choice of a cell size is critical: bigger cells decrease their total number and the algorithm performs faster but, on the other hand, we lose the resolution of the environment. A properly fixed cell size for the GPP should be in the order of dimension of the robot, or somewhat smaller.

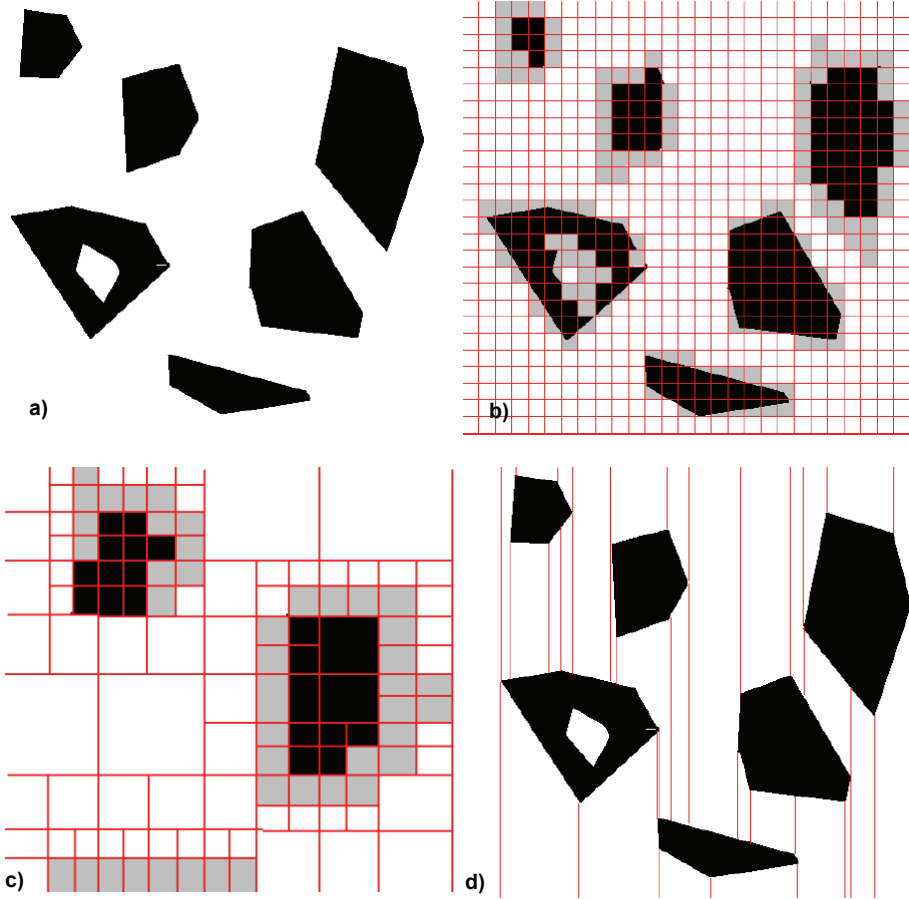


Fig. 3. Cell decomposition for the metric map: a) the binary base metric map, b) the approximate cell decomposition of a cell larger than the base one, c) the adapting cell decomposition, and d) the exact vertical cell decomposition: white – a free cell, black – an occupied cell, gray – a partially occupied cell.

Three main types of cell decomposition can be found in the literature: (1) approximate, (2) adapting, and (3) exact (cf. Fig. 3). In the *approximate decomposition* one constructs a set of non-overlapping cells that entirely covers the C-space. It is assumed that each cell has a simple, standard shape that implies the ease of

computation. A regular shape of a cell is also important for checking the adjacency of cells by finding cells with common borders. The most common approximate cell decomposition method is the *rectangloid decomposition*.

If we assume m as the dimension of the C-space C and M a closed rectangloid in $C = R^m$, then we define approximate cell decomposition D_A as a finite set of non-overlapping cells c_i so that:

$$M = \bigcup_i c_i \quad (1)$$

$$\forall i \neq j \quad c_i \cap c_j = \emptyset \quad (2)$$

If one denotes by $C_0 \subseteq C$ a rectangloid of C that is occupied by obstacles and by $I(c)$ the interior part of cell c , one can assign a label (value) to each cell c_i from the C-space as follows:

- Occupied: if $I(c) \subseteq C_0$,
- Free: if $I(c) \cap C_0 = \emptyset$, $I(c) \subseteq C_{free}$,
- Partially occupied: if $I(c) \cap C_0 \neq \emptyset \wedge I(c) \cup C_0 \neq C_0$.

A channel is a sequence of adjacent cells in D_A containing only free or partially occupied cells [40]. The channel is called an *F-channel* if and only if all cells in the sequence are free; otherwise one speaks about a *P-channel*. In the light of that, the GPP problem is to find a channel between the initial and destination configuration.

A great advantage of the approximate cell decomposition is that it provides a simple and uniform tool for sampling the region M of C that is easy to implement. One can also set a desired granularity corresponding to the size of c resulting in some form of adaption. On the other hand, the decomposition is not based on C_0 and therefore D_A suffers from:

- redundancy of information in the form of adjacent free cells or adjacent occupied cells,
- lack of appropriate information in the form of assigning a partially occupied value.

The graph methods based on the cell decomposition use the middle points of adjacent cell borders for creating the vertices.

On the other hand, we deal with the *adaptive cell decomposition* (Fig. 4) by performing D_A for partially occupied cells iteratively until the basic grain or a fixed resolution ε of map is reached or no partially occupied cells exist on M . Therefore, M in the adaptive cell decomposition is decomposed by $D_A = D_{A0}, D_{A1}, \dots, D_{Ai}$, i.e. via the recursive rectangloid decompositions. If ε is chosen to be small enough, then all *P-Channels* are changed into the *F-Channels* (if such exist). As the number of decomposition steps depends on the content of M , the computation time increases in the worst case by four times each next decomposition is performed. The method of decomposing each partially occupied cell into a set of smaller non-overlapping cells of a rectangloid shape is a hierarchical process and can be described in terms of three decomposition levels (cf. Fig. 4).

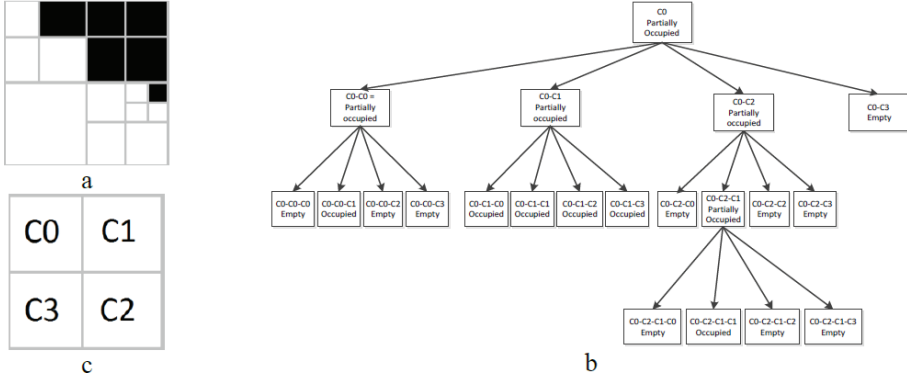


Fig. 4. An example of the hierarchical adaptive cell decomposition for $m = 2$: a) M on which the decomposition is performed, b) the decomposition tree, c) the cell numbering convention used. The graph after three layer decomposition is build up of 16 cells in the adaptive decomposition. In the approximate decomposition method one would have 64 cells assuming the third layer granularity decomposition.

The degree T of each of the following decompositions $D_{A0}, D_{A1}, \dots, D_{Ai}$ is, in the case of the rectangloid cell shape, always $T = 2^m$.

The exact cell decomposition methods do not provide a regular grid cell but rather a set of cells – still none overlapping – that divide M based on the content of M . Therefore, the cells are polygons of different shape and size. This ensures no redundancy in the free space description but requires a more complicated single cell description. The exact cell decomposition methods fall into the following main types, depending on how each cell is constructed:

- *Trapezoidal Decomposition* (Fig. 5a). In this approach all closed objects belonging to C_0 (obstacles) are approximated by polygons (see Fig. 5). Each polygon vertex v becomes a starting point for a vertical (horizontal) segment connecting v with the borders of M or edge of other object of C_0 (red in Fig. 5a). As all segments from all vertices v are parallel, then a cell has the form of a trapezoid. The middle points of each segment become the vertices in the graph.
- *Cylindrical Decomposition* (Fig. 5b). This approach is very similar to the former one except for one fact: if a segment ends on another polygon, then it is extended until it reaches the border of M ; the middle points of each segment become the vertices in the graph.
- *Triangulation* (Fig. 5c). In this approach each vertex of polygonal objects in C_0 is connected with two other ones so that the cells are of a triangular shape.

In the exact decomposition the number of graph vertices depends on the content of M . The critical issue when designing a graph based GPP algorithm is a proper choice of vertices – even the best graph search algorithm will be useless if no optimum connections between the vertices can be found².

² Refer to Fig. 5a, b, c, when in case a, b there is no way that allows to circle pentagon while a more appropriate choice of vertices in case c provides a way around this difficulty.

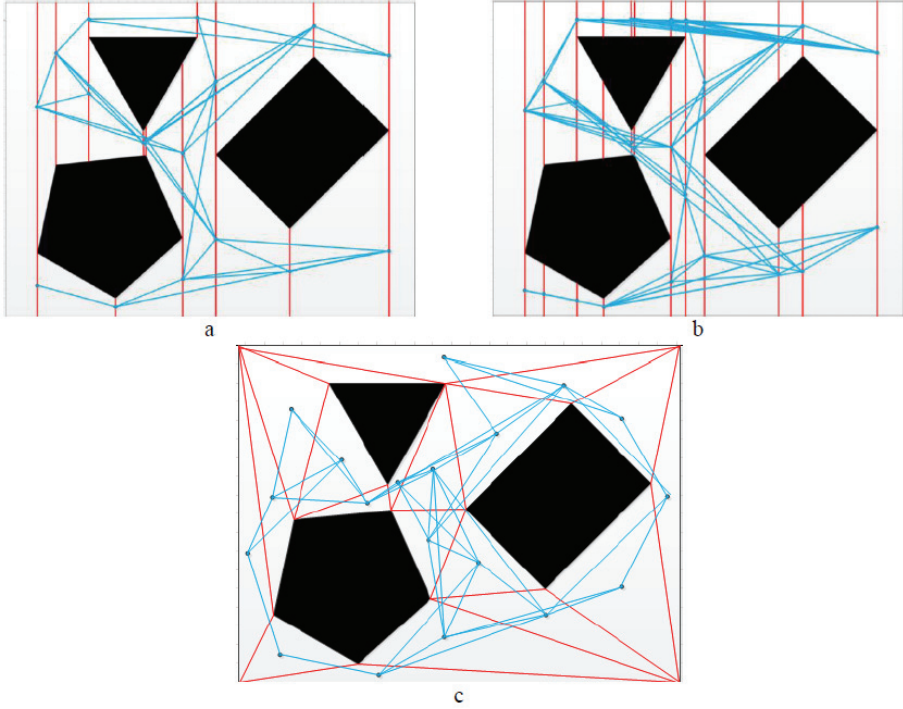


Fig. 5. An example of the exact cell decomposition. a) the trapezoidal decomposition, b) the cylindrical decomposition, c) the triangulation decomposition; white – free space, black – occupied space, – – decomposition orders, – – possible inter-cell crossing.

2.6. Mathematical formulation of the problem

Independently of the type of the S-space, map, and C-space, the GPP problem, denoted generally as $P(x)$, where x is a particular instance of the problem, can be described as a vector (notice that we will use in the explanation of the particular elements of the vector those terms which directly refer to an evolutionary computation related view of the problem and its solution as it will be clear later on):

$$P(x) = \langle C, S, A, A_R, N, M, B_{GPP}, E_{GPP}, g, f, \varphi, r_0, \dots, r_m \rangle \quad (3)$$

- C – Configuration space (C-space) of the problem – the space of genotype elements of problem P ,
- S – Search space (S-space) of the problem – the genotype space of problem P ,
- A – Answer space for problem P – the phenotype space of problem P ,
- N – a neighbourhood function,
- A_R – a set of solutions for problem P , $A_R \subseteq A$,
- m – a map representation of the C-space,
- B_{GPP}, E_{GPP} – an initial and destination position of the path, $B, E \in M$,
- f – an evaluation function which determines the "goodness" of a path,

g – a growth function: *each* element $s \in S$, called a candidate solution, represents at least one answer $a \in A$ for problem P , which is done through the growth function g :

$$g : S \rightarrow A \quad (4)$$

$r_0, \dots, r_m$ – planning rules which determine the feasibility of a computed path,

φ – a feasibility function for a path that in case of the GPP problem has the form of the Boolean conjunction:

$$\varphi(a) = \varphi_b(a) \wedge \varphi_e(a) \wedge \varphi_r(a) \quad (5)$$

$a \in A$ – a particular candidate answer (path of a mobile robot) for problem P , i.e.

$$a = (Q_0, \dots, Q_{L(a)-1}) \forall i < L(a) - 1, Q_i \in M, a \in A \quad (6)$$

$$\varphi_b(a) = \begin{cases} 0 & \text{for } a[0] \neq B_{GPP} \\ 1 & \text{for } a[0] = B_{GPP} \end{cases} \quad (7)$$

$$\varphi_e(a) = \begin{cases} 0 & \text{for } a[L(a)-1] \neq E_{GPP} \\ 1 & \text{for } a[L(a)-1] = E_{GPP} \end{cases} \quad (8)$$

$$\varphi_r(a) = \bigwedge_{j=0}^m \varphi_{r_j}(a) \quad (9)$$

$$\varphi_{r_j}(a) = \begin{cases} 1 & \text{for a candidate answer not violating rule } r_j \\ 0 & \text{for a candidate answer violating rule } r_j \end{cases} \quad (10)$$

$$a \in A_R \Leftrightarrow \varphi_r(a) = 1 \quad (11)$$

The A_R set is not explicit but the belongingness $a \in A_R$ is determined for each candidate answer a by the set of rules r_j . $L(a)$ is the number of intermediate points within the path? The minimum value of $L(a)$, $\forall a \in A$, is 2, because each path contains at least the initial and final points, B_{GPP} and E_{GPP} .

The fulfillment of $\varphi_b(a) \wedge \varphi_e(a)$ does not pose any difficulty, while the case of not violating any rule of path planning, $\varphi_r(a) = 1$, is a hard task; the latter (i.e. $\varphi_r(a) = 1$) is known in computer science as the Boolean Satisfiability Problem (SAT) that is an NP-complete problem.

This implies that the GPP problem described in the form (3) is a SAT-type problem apart from being an optimization problem. If $A_R \neq A^3$ for a particular instance x of problem P , then the transformation (12) proceeds. Otherwise ($A_R = A$), the transformation (12) is not performed on the x instance of problem P . The inequality $A_R \neq A$ holds if and only if $\exists a \in A, \exists j$, that: $\varphi_{r_j}(a) = 0$. Then:

³ Meaning that all possible paths do not violate any planning rule.

$$\begin{aligned} & \langle C, S, A, A_R, N, M, B_{GPP}, E_{GPP}, g, f, \varphi, r_0, \dots, r_m \rangle \\ & \rightarrow \langle C, S, A, A_R, N, M, B_{GPP}, E_{GPP}, g, f', \varphi', r_0, \dots, r_m \rangle \end{aligned} \quad (12)$$

$$\varphi'(a) = \varphi_b(a) \wedge \varphi_e(a) \quad (13)$$

$$f'(a) = f(a) + \sum_{j=0}^m w_{r_j} \cdot q_{r_j}(a) \quad (14)$$

w_r – normalized weighting factors for rule violation functions, $\forall r \ 1 \geq w_r \geq 0$,
 $q_{r_j}(S)$ – a rule violation function for a particular path over map M that has the following important properties:

$$\begin{aligned} & \forall a_1 \in A, a_2 \in A, [\exists j : \varphi_{r_j}(a_1) = 0 \wedge \varphi_{r_j}(a_2) = 1] \wedge \\ & \wedge [\exists i \neq j : \varphi_{r_i}(a_1) = 1 \wedge \varphi_{r_i}(a_2) = 0] : f'(a_1) < f'(a_2) \end{aligned} \quad (15)$$

$$\forall a \in A, \forall j : \varphi_{r_j}(a_1) = 0 \Rightarrow q_{r_j}(a_1) = 0, \quad \forall j : \varphi_{r_j}(a_1) = 1 \Rightarrow q_{r_j}(a_1) > 0 \quad (16)$$

Properties (15) and (16) are crucial for the construction of the rule violation functions $q_{r_j}(S)$ and can also be expressed by:

$$\forall a_1 \in A_R, \forall a_2 \in A - A_R \ f'(a_1) < f'(a_2) \quad (17)$$

As it will be shown, the inequality $f'(a_1) < f'(a_2)$ in (15) has to be replaced by $f'(a_1) \ll f'(a_2)$ in order to reach a fast convergence to a violation-free path of a mobile robot in the case of using a memetic algorithm as it will be presented in this work later on.

The weighted sum $\sum_{j=0}^m w_{r_j} \cdot q_{r_j}(a)$ in (14) expresses a distance of feasibility, i.e. a distance of a to A_R . For the problem described by (12) and (3), the GPP is an optimization problem with constraints and φ_{r_j} are the Boolean feasibility functions. In case of both (3) and (12) the constraints are set by rules r_j , which are explicitly given (3) before the GPP is undertaken. These rules may include: a necessity to omit obstacles, a necessity not to get closer to an obstacle than a specified distance, a necessity of avoiding movement loops, etc.

In the case of a one-dimensional function $f(s)$, the problem of GPP is a single-objective one and can be solved by using typical optimization methods. If there are several objectives that one has to incorporate, the problem becomes practically solvable only via approximate (heuristic) methods.

A neighbouring function N is defined as:

$$N : A \rightarrow 2^A : N(a) \subseteq A \quad (18)$$

and is a function for selecting neighbouring configurations of problem P . It defines possible transitions between the configurations. The N -function defines the “locality” of a local search algorithm incorporated to find solutions $a \in A_R$. The principal property of N is that for each pair of configurations $a_1, a_2 \in S$ there is a finite number (here n) of neighbouring configurations $a'_1, \dots, a'_n$, such that:

$$a'_1 \in N(a_1), a'_2 \in N(a'_1), a'_3 \in N(a'_2), \dots, a'_n \in N(a_2) \quad (19)$$

i.e. there is a finite number of “steps” between each two candidate solution; in other words, each candidate solution can be created from any other configuration from S .

Using this notion, we have:

The Global Path Planning problem P is a problem of finding a feasible answer a^* such that:

$$[a^* \in A_R : \forall a \in A f'(a^*) \leq f'(a)] \Leftrightarrow A_R \neq \emptyset$$

If A_R is an empty set, then any GPP algorithm should output an error informing that no valid path can be found.

In general there are two different types of approaches to each search/optimization problem [41]: a *Local Based Search* and a *Population Based Search*. The Local Based Search methods start from a candidate solution a_0 and search its $N(a_0)$ for an a_1 candidate solution which leads to a better value of the evaluation function f . Because of property (19) all the configurations can be evaluated. The main property of the Local Based Search is the selection of a neighbouring function N and performing a “step” from one configuration to another within neighbourhood N . As the “step” operation is always just one step, then the Local Based Search boils down to an iterative mutation process as it will be clear later on.

On the other hand, we can well imagine an immediate expansion of the idea of the *Local Based Search*, i.e. the *Population Based Search* when one keeps n ($n \geq 2$) concurrent candidate solutions. In this manner not only one can speed up the computation process by some parallelization but the *Population Based Search* brings about another improvement in the evaluation of the f' function: a *recombination operator*. This operator combines several configurations into a new configuration. As some candidate solutions are independent from each other, one can benefit from the parallel and concurrent computations. The candidate solutions cooperate and compete between each other. The aim of doing so is to overcome shortcomings of each candidate solutions and amplify their merits.

In case of the Population Based Search we no longer speak about a simple point within the S -space but rather about a hypergraph in that space [41], which tries to cover that space in the best possible uniform manner.

The resulting conclusion that a Global Path Planning is an optimization problem with constraints is crucial. It means that nearly all type of the variety of optimization methods present in the literature may be employed to solve this task. Therefore one should expect a diversity of methods solving this problem.

3. Classical methods for solving GPP

Depending on the world representation (map) methods the user has to choose a proper way to solve the GPP problem. The methods differ from each other and it is really difficult to compare one method to another and they should rather be compared within the group of methods that they belong to. A great part of the GPP methods are based on some random processes [31] and the properties of time and space complexity cannot be easily or explicitly calculated for them.

The path optimality is one of the key feature of a desired algorithm and most of methods deliver it though only within the representation employed. However, in general, not necessarily an “optimal path” is the best path.

The methods are distinguished by the map representation type they use starting from computational geometry based methods through roadmaps and graph solving techniques and ending in meta-heuristic based approaches. Such an order is based in fact on the chronology of their appearance.

The GPP algorithm is called *complete* if and only if it returns a valid path between the start and destination point, if it exists, but also indicates if no solution in the search space exists that does not violate planning rules. In general, theoretical, deterministic algorithms lack the property of completeness, they are difficult to implement and are not robust. On the other hand, heuristic algorithms do not guarantee the attainment of an optimal solution but they usually give good results and are easier to develop and implement.

3.1. Roadmap methods

3.1.1. Visibility graph methods

In the early 1980's one of the most important problem in computational geometry that was dealt with computationally, and the solution of which found many applications, was the problem of finding obstacle avoiding shortest paths [42] and of finding efficient algorithms for the visibility graphs (the visibility graphs and the obstacle avoiding shortest paths are described in [32, 33]). It can be shown that a path between two points is the shortest in the sense of its Euclidean length if it is a spline of line segments whose vertices are vertices of obstacles. The graph that connects the vertices V of polygonal shape obstacles is called a “visibility graph” an example of which is shown in Fig. 6.

The *visibility graph* is an undirected graph with the set of vertices V and the set of edges E connecting the vertices. For constructing such a graph there is a straightforward method: for each pair of vertices, p, q , one checks if a line segment connecting them intersects with any obstacle. If $|V| = |E| = n$, then the time complexity of such an algorithm is $O(n^3)$.

For the first time the algorithm was presented by Lozano-Perez and Wesley [23] with the computational complexity of $O(n^3)$. Then, by introducing theoretical and algorithm improvements, that complexity was consecutively reduced to

$O(n^2 \log n)$ [24, 43], $O(n^2)$ [44], $O(|E| \log n)$ [45] ($|E|$ is the number of edges in the visibility graph), and $O(n \log n + |E|)$ [46, 47].

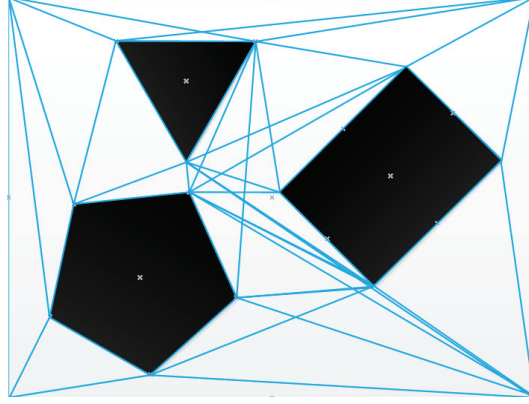


Fig. 6. An example of a visibility graph. Note that each edge of a polygonal obstacle becomes an edge in the visibility graph. V is completed by the map region of the internal vertices.

In general an approach for reducing the computational complexity consists in using different approaches for finding the visibilities. Assuming V to be a set of vertices of polygonal obstacles and E a set of edges of these obstacle, we note that each vertex $v \in V$ is a point of connection of two segments $e_{\text{segin}}(p)$, $e_{\text{segout}}(p)$ (see Fig. 7). The $\text{ray}_d(p)$ is a half line with p as the starting point and d as the direction.

A visibility function $\text{vis}_d(p)$ in the direction connected with d is defined as:

$$\text{vis}_d(p) = \begin{cases} \infty, & \text{if there is obstacle in direction } d \\ \text{INTERNAL}, & \text{if half line } d \text{ intersects polygon of which } p \text{ is a vertex} \\ e, & e \in E \text{ in other cases} \end{cases} \quad (20)$$

The algorithms of constructing the visibility graph starts from $d = 0$ (the horizontal direction, see Fig. 7). Then, one begins changing the direction d from 0 to π by simultaneously updating the value of vis_d for each vertex of the polygon. Denoting by d_{prev} a half-line preceding d , and by d_{follow} the direction following d , we update according vis_d to:

Step 1. If $\text{vis}_{d_{\text{prev}}}(p) = \text{INTERNAL}$ and point q is the other endpoint of $e_{\text{segout}}(p)$ or if $\text{vis}_{d_{\text{prev}}}(p) \neq \text{INTERNAL}$ and q is an endpoint of $\text{vis}_{d_{\text{prev}}}(p)$, then

$$\text{vis}_{d_{\text{follow}}}(p) = \begin{cases} e_{\text{segout}}(q) & \text{if } \text{vis}_{d_{\text{follow}}}(q) = \text{INTERNAL} \\ \text{vis}_{d_{\text{prev}}}(q) & \text{in other cases} \end{cases}$$

Step 2. If $\text{vis}_{d_{\text{prev}}}(p) \neq \text{INTERNAL}$ and q is closer than $\text{vis}_{d_{\text{prev}}}(p)$ in direction d , then

$$\text{vis}_{d_{\text{follow}}}(p) = e_{\text{segout}}(q)$$

Step 3. If $vis_{d_{prev}}(p) \neq INTERNAL$ and q is behind $vis_{d_{prev}}(p)$ in direction d or if $vis_{d_{prev}}(p) = INTERNAL$ and q is not the other endpoint of $e_{segout}(p)$, then no changes are made to $vis_{d_{follow}}(p)$ and $vis_d(p)$.

If $vis_{d=0}(p)$, then the calculation is performed for each $p \in V$, one sorts $\binom{n}{2}$ pairs of vertices by the slope of their connecting lines and for every pair of vertices, vis_d is calculated according to Steps 1–3 mentioned above. The complexity of the $vis_{d=0}(p)$ calculation is $O(n^2)$, the sorting by using well-known procedures can be performed in $O(n^2 \log n)$ and totally the complexity is of $O(n^2 \log n)$.

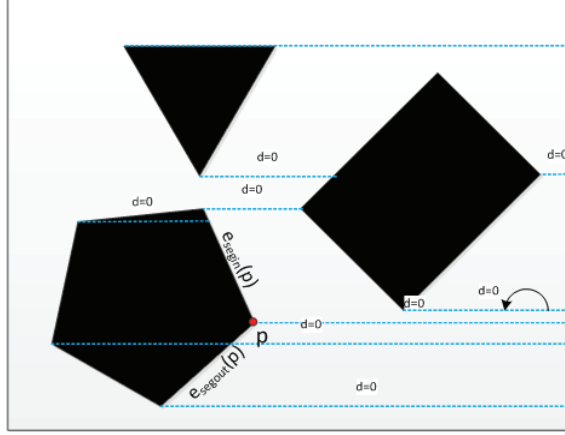


Fig. 7. An example for the calculation of an existent visibility graph in $O(n^2 \log n)$.

Another way to decrease the time complexity of calculating the visibility graph is to construct a sparse visibility graph. This method was first presented by Ghosh and Mount [46] and further developed by Rohnert [48] and Tarjan [49].

The visibility graph can be used to determine all the shortest paths to the destination (endpoint). Vertices in the visibility graph are completed by the starting and destination points for which the visibilities are also calculated. Depending on the implementation of the Dijkstra algorithm [50], we need to increase the time complexity by $O(n^2)$. (the classic implementation) or by $O(|E| + n \log n)$. (the Fibonacci heaps based implementation [51]).

3.1.2. Shortest path map methods

This group of methods is also meant for polygonal obstacles. Map M is divided into regions $R_1, R_2, \dots, R_n$. We say that two points p, q belong to the same region R_i if:

$$p, q \in R_i \Leftrightarrow \forall w \in M \pi(p, w) = \pi(r, w) \quad (21)$$

where $\pi(a, b)$ is the shortest path between points $a, b \in M$ excluding the start and destination points.

This means that two points belonging to the same R_i have exactly the same routing topology from any point in the map. Therefore, in order to find the shortest path between any pair of points from M one needs to define all regions R_i and determine the position of the starting and destination points to the particular regions. Effective and efficient point location methods can be found in [52, 53].

Another example of the shortest path maps was discussed by Lee and Preparata [54]. They attained the $O(n \log n)$ time complexity if all obstacles were parallel line segments. Other examples can be found in [55] (with $O(n(k + \log n))$, where k is the number of obstacles), in [56] ($O(n \log^2 n)$ when the visibility graph is used for the determination of vertices and graph edges).

3.1.3. Shortest path methods using triangulation

Assuming the GPP task within a simple polygon P , another group of methods was introduced by Lee and Preparata [54]. The first step of these methods consists in the triangulation of the interior part of the polygon within which a path is planned. The triangulation that is asymptotically fastest is that presented in Tarjan and Wyk [49]. Vertices V of graph T are constructed as cross points of the medians of each triangle, and edges of the graph are line segments with neighboring triangles, i.e. the maximum number of edges starting from any point of such a graph is tree.

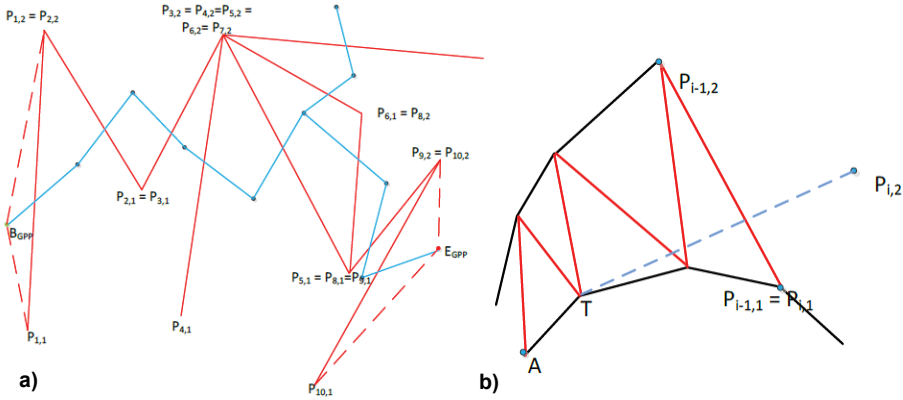


Fig. 8. An example of the GPP within a simple polygon, B_{GPP} – starting point, E_{GPP} – destination point; a) polygon P after the triangulation and tree graph determination, b) polygon after reducing the cells $C(B_{GPP})$, $C(E_{GPP})$.

In the next step, polygon P is reduced by downsizing the cells $C(B_{GPP})$, $C(E_{GPP})$ containing the start and destination points. Each cell border is a line segments $|P_{i,1}, P_{i,2}|$ (Fig. 8). The shortest path from the P_i and B_{GPP} , E_{GPP} points is a line segment connecting these two as there is always visibility between them. If $\pi(B_{GPP}, P_{i,1})$ and $\pi(B_{GPP}, P_{i,2})$ is a graph trace from B_{GPP} to $P_{i,1}$ and from B_{GPP} to $P_{i,2}$, respectively, and the last common point for these two paths is point A (see Fig. 8b), the segments of $\pi(B_{GPP}, P_{i,1})$, $\pi(B_{GPP}, P_{i,2})$ are convex within P which is called a *funnel* [54]. This common part of a path becomes a destination one. The remaining sequence of vertices to visit is calculated as follows:

Step 1. If $A, P_{i,2}$ is a visibility, then it becomes a parent of a path.

Step 2. Otherwise, start by a sequential search among the vertices in the funnel F for a vertex T such that the line segment $|T, P_i|$ is tangent to F . A polygonal line connecting A and T becomes the new funnel F .

The class of algorithms that find the shortest path in $O(n \log \log n)$ may be further improved by triangulation to $O(n)$ [54].

The main drawback of the method presented is that it works only for the interior of polygons. In the case of rectangloids, the triangulation dividing area is very sparse resulting in a non-optimal selection of graph vertices. Each of the method mentioned before are based on the old Lozano-Perez and Wesley approach [23] and can deal with a point robot. Each real non-dimensionless robot demands a map dilation. In the case of a non-polygonal shape of the robot or the kinematic structure corresponding to the track based robot steering, the dilation results in a change of polygonal-shaped obstacles into non-polygonal ones, and therefore all the methods listed cannot be applied.

A very similar idea of a roadmap is called a *meadow map* [57, 58] an example of which is shown in Fig. 9. As in the case of triangulation, the path planned is jagged and not optimal in the sense of length. For this method a technique of path relaxation has been developed that is based on the construction of a path produced by the meadow map method which calculates a path similar to the visibility diagram path (see Fig. 9).

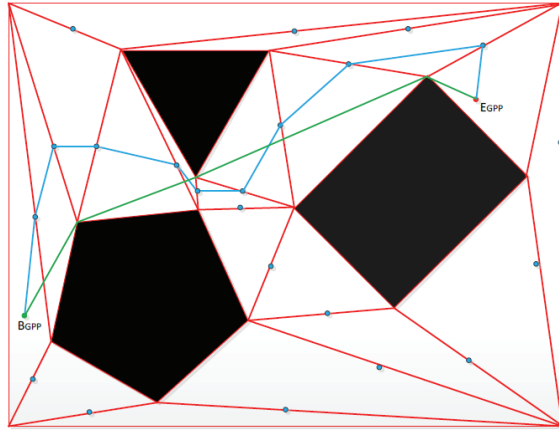


Fig. 9. An example of the *meadow map*. Vertices of the graph constructed are marked with the blue dots: the middle points of the red segments connecting vertices of each polygon. The blue line is the meadow map path and the green line is the meadow map path after relaxation.

Chew [59] in his work presented a visibility graph approach for an ellipsoidal shaped robot that runs in $O(n^2 \log n)$. A similar work can be found in [60]. Enriching Chew's algorithm by using the *SPM* approach, the path-planning for an ellipsoidal robot can be performed in $O(n^2)$ [55].

3.1.4. Voronoi diagrams

The Voronoi diagram is a special case of the metric cell decomposition [61]. In most of the examples considered it divides the plane represented by M containing n points $O_0, \dots, O_{n-1}$ to n non-overlapping cells $C_0, \dots, C_{n-1}$ so that any point $P \in C_i, P \in M$, is closer to O_i than to any other point $O_0, \dots, O_{n-1}$:

$$\begin{aligned} \forall M : O_0, \dots, O_n \in M \wedge \bigcup_{k=0}^{n-1} O_k = C_0 \xrightarrow{\text{def}} M = [C_0 \cup \dots \cup C_{n-1}] \wedge \\ \wedge (\forall i \neq j : C_i \cap C_j) \wedge [\forall P \in C_i, \nexists j < n : d(P, O_i) > d(P, O_j)] \end{aligned} \quad (22)$$

d – a distance function between any two points within M .

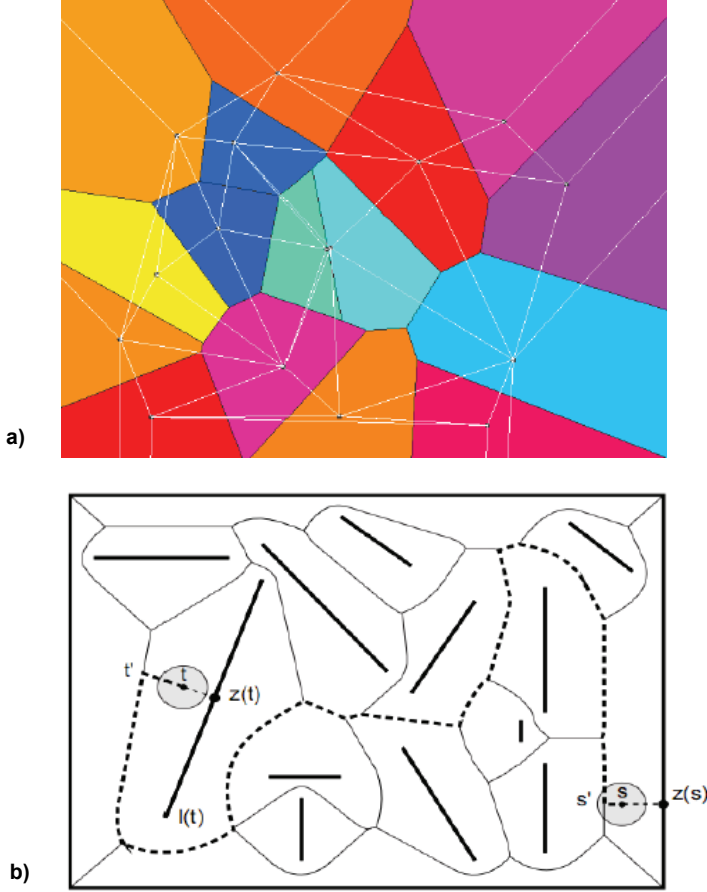


Fig. 10. a) An example of the Voronoi diagram cell decomposition derived by using a dedicated tool [69]. Different colors refer to different cells $C_0, \dots, C_n$. The black dots denote $O_0, \dots, O_n$ while the white segments are the Delaunay edges [63]. b) the Voronoi diagram for the line segments [61].

The edges of $C_0, \dots, C_n$ become the edges of the graph. The vertices are constructed as points of the edge intersection. Due to the formal definition (22), the graph edges that represent possible pathways between the vertices maximize the distance of the robot to obstacles while following the path. An example of the Voronoi diagram is shown in Fig. 10.

The lower bound for computational complexity for the Voronoi diagram was established by Shamos [64] and is at least equal to $O(n \log n)$ for n points, the coordinates of which are increasing.

Several approaches were proposed to construct the Voronoi diagrams:

- *Incremental construction* [65]. In this natural approach $V(S)$ is constructed by incrementally adding points, i.e. for point p , $V(S)$ is derived from $V(S)/p$ by inserting this point. The computational complexity of such a procedure is $O(n^2)$. Some related works that make it possible to lower this bound can be found in [66, 67].
- *Divide and conquer method* [68]. In this approach the area S is divided by vertical or horizontal lines into two regions L and R for which $V(L)$ and $V(R)$ are calculated recursively. If the splitting lines between points can be found in $O(n)$, then this method attains the $O(n \log n)$ complexity.

In real world conditions point obstacles occur rarely, if at all, and the definition (22) has to be extended to polygonal shape obstacles. This consists in finding polygon edges and representing them as line segments, and then in performing the bisection of the regions [63] – see Fig. 10b.

For a disc shape non-point robot the collision-free path between any two points on the map exists if and only if the radius of the disc is not larger than any point of the path along the Voronoi diagram. The proof of this theorem can be found in [61]. The assumption of a disc-shaped robot is a typical approach to most of the robot shapes because by using this approach one can also model the kinematics of motion for a track based robot that includes turning possibilities.

3.1.5. Probabilistic roadmaps

The *Probabilistic Roadmap Planning* (PRM) is a relatively a new technique in the straight graph approach that has been developed over recent years by different researchers [69–73]. The family of *PRM* methods has been proved to be successful in many types of motion planning tasks. Generally speaking, the *PRM* consists in two recursively performed steps: (1) sampling the C-space, and (2) building a graph by simple planners. There is no one and only one specific method for these steps but one has to rather choose a specific one that will perform best in a particular case considered.

The C-space is sampled for some configurations c and if $c \in C_{free}$, then this configuration is added to graph V , and then a set of vertices from the existing V is tested for a collision free connection with c and a new edge in V is created, if it exists. Note that, in general, not all vertices from V are checked for that connection but rather a number of them is sorted increasingly with respect to the distance from c , so that the edges in V remain local. Several approaches to the sampling can be found in the literature:

- *Uniform sampling*, in which the density of candidate vertices is constant within C_{free} . This is also called a *grid based sampling*. Typically, a grid is coarse at the first time and is refined by making the grid denser if no connection can be made.
- *Random sampling*, when a random sample is created by picking-up random values for all degree of freedom of the C-space.
- *Halton sampling* [74] or *random Halton sampling*. The C-space is sampled due to the Halton discrepancy theory for achieving a better coverage than the grid based sampling. For more details on the Halton sampling, cf. [75]. In the random Halton approach, while adding the next n -th vertical candidate, one checks the area in the neighborhood of this point for the next vertical candidate and the neighborhood is limited to:

$$N = \frac{kA}{n} \quad (23)$$

A – an initial part of the C-space (a significant part of it),

k – a small constant; typically $k \approx 0.002$ [76],

n – the number of the next candidate for vertical,

so that each next point sampled in N becomes smaller.

- *Cell-based sampling*. This is an iterative method. One starts by decomposing the workspace into 2^n cells and assigning to them random configurations. Next, each of these configurations are split into 2^n subcells. This method refers to an approximate cell decomposition. In this way we attain a better spread over C_{free} .
- *Gaussian sampling*. The main purpose of a diversification of the density of sampling in different regions of the C-space is to add more configurations near the obstacles. It is done by sampling two configurations that lie apart from each other by the distance of the Gaussian distribution simultaneously and adding a non-violating one of them only if the second lies in C_0 . It was shown in [77] that the Gaussian sampling shows excellent performance for indoor conditions.
- *Obstacle-based sampling* [78] works in a similar manner. If a sampled configuration lies in C_{free} , then it is added to the graph, otherwise a random direction of the movement is chosen and a next point is sampled with an increasing step until the sampled point belongs to C_{free} . In this way the sample configuration lies near the boundary of C_{free} and C_0 . Another version of the obstacle-based approach (called the obstacle based* approach) refrains from adding a point to the graph if initially it was in the C_{free} region leading to a situation that most of the sampled points are close to the $C_{free} - C_0$ boundary.

Geraerts and Overmars in [76] present the total time of sampling around obstacles for the methods mentioned while used in the PRMs for several different indoor terrains. Their results are shown in Table 1.

The second important step in PRM is *node adding*. When the vertices of the graph are found, they have to be incorporated in the graph by setting a neighborhood

relationship between the vertices. A proper choice of n , the maximum number of neighboring vertices, is a key point from the point of view of time complexity of the connectivity computation. Too few point may lead to the situation when no graph connection can be found, on the other hand the connection of the vertical ones that are far away from each other in the C-space results in a high computation efficiency because a test for a connection is usually the most expensive part in PRM [78].

Table 1. A comparison of the computation time (in seconds) of sampling methods used in the probabilistic roadmap approach (PRM) – cf. Geraerts and Overmars [76].

Sampling Area	Gaussian	Obstacle based	Obstacle based*	Halton random	Halton deterministic	Random	Cell based	Grid based
Corridor	0.4	0.5	0.5	0.3	0.2	0.4	0.4	1.4
Rooms	0.4	0.7	0.5	0.2	0.3	0.8	0.7	0.6
Clutter	5.6	3.7	7.9	2.2	4.0	2.7	3.1	7.3
Hole	3.1	8.0	2.1	15.8	43.3	33.9	42.3	26.2
House	268.0	197.1	209.4	152.9	207.4	210.0	225.9	262.5

As in general the C-space is more than two dimensional and the robot is treated as non-holonomic, then the difference in two configuration is not only to be determined by the Euclidean distance but also a rotational distance has to be taken into account. It is done in most cases by summing up two distances $d = d_t + d_r$, where d_t denotes a translational distance between the configurations and d_r denotes a maximum distance that each of the points belonging to robot travels while the robot rotates, i.e. the point that lies farthest from the rotation origin. While this method is easy to compute, it also gives a reasonable estimate of the configuration distance [31].

The following techniques are used for the determination of graph edges:

- *Nearest- n* . In this approach only n vertices laying the closest to configuration c are added. Close vertices result in an efficient connection computation. A drawback of this method is that if a component (obstacle) has multiple vertices all n -nearest vertices are associated with the same component. Therefore no intra-component connections are made and no connection through the graph can be found.
- *Component*. Each configuration is connected to a component that lies close enough as it is preferable to connect multiple components. An N -component version connects configuration c to at most n configurations from each component. The number of connections is kept small as in the nearest- n case.
- *Visibility*. The approach is based on the configuration visibility [79] that connects useful vertices. A candidate vertex is useful if it cannot be connected to any other vertex or to any other two or more vertices. If the visibility is from one other vertex only, then this configuration is not useful.

It is showed in [76] that depending on the terrain type, different vertex addition methods may be preferable. Results of the comparison of edge building strategies can be found in Table 2.

The PRM method requires a proper choice of the sampling and edge adding technique. The best results can be obtained by combining the Halton sampling with the nearest- n edge adding strategy; Table 1 and Table 2 can be used for comparison,

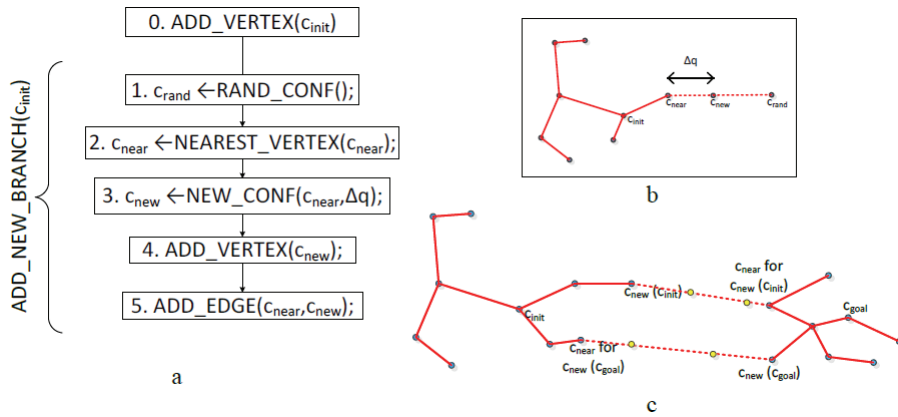
Table 2. A comparison of computation time of vertex adding strategies for the PRM – cf. Geraerts and Overmars [76]⁴.

Area	Nearest- n	Component	Component- n	Visibility
Corridor	0.2	0.8	0.5	1.5
Rooms	0.3	0.5	0.7	2.3
Clutter	4.0	4.4	3.4	7.6
Hole	43.3	120	31.9	20.9
House	207.4	279.6	189.1	> 600

3.1.6. Rapidly exploring random trees

Another randomized approach to the GPP problem was introduced by LaValle [80]. It is a graph based method that can be used for the holonomic, non-holonomic [81] and kinodynamic planning [82] of moving objects including the mobile robots [83].

In general, the method for the constructing of a random tree of n vertices rooted in c_{init} is shown in Fig. 11. First, an initial configuration is added to the graph, and the next steps are repeated n times to obtain an n -vertex graph.

**Fig. 11.** An example of the RRT [91] (Rapidly exploring random trees) method: a) a general algorithm, b) a vertex addition procedure, c) an example sequence of two rapidly exploring random trees (RRTs) rooted in c_{init} and c_{goal} which build up the connection graph.

A random configuration is sampled from the C-space as follows: c_{rand} (step 1) $c_{rand} \in c_{free}$. In step 2 c_{near} is found, i.e. a vertex in the configuration graph for which the distance from c_{rand} is the closest among all the added ones. The c_{rand} and c_{near} create a segment in which a new graph vertex is created, c_{new} : that is placed in the direction of c_{rand} from c_{near} by the Δq distance. The RRT graph is built from the c_{init} and c_{goal} configurations simultaneously and the c_{near} vertices are sampled from both roots.

⁴ The computation time is provided for a typical desktop computer with a 2.4 GHz Pentium 4 1 GB RAM.

The `ADD_NEW_BRANCH` procedure is calculated for c_{init} and c_{goal} , simultaneously for c_{new} , obtained from c_{init} , and c_{new} , obtained from c_{goal} if there is visibility between c_{new} and c_{near} for the goal and initial point (see Fig. 11c); two roots are connected and a path is found. To connect two roots, the rule is assumed that additional vertices in the C-space with a distance of Δq are produced using a greedy algorithm and therefore the path produced need not be optimal and the algorithm is not complete. However, as in the case of the PRM, it can be shown that it is probabilistically complete.

The main issue that is responsible for time complexity of the RRT method is the search for the nearest neighbor. For trees with more than 20 000 nodes a typical brute-force method for finding the nearest neighbor has to be replaced with more sophisticated counterparts such as the k-d trees [85] or a bi-objective shortest path method [86]. An example of a path planned using the RRT algorithm can be found in Fig. 12. Additional information on the RRT methods can be found in [84, 87, 88].

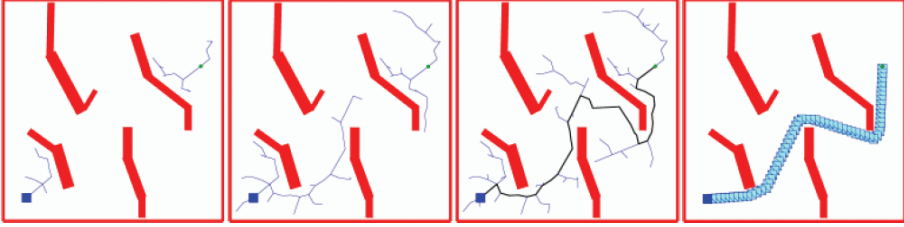


Fig. 12. An example of a path planned using the RRT algorithm c.f. [86].

3.2. Artificial Potential Fields methods

3.2.1. Artificial Potential Fields

The *artificial potential field method*, first introduced by Khatib in 1986 [89], was originally developed as a real-time obstacle avoidance method when emphasis was put on online computations. In this approach a robot is treated as an electron in a potential field on the map where attractive and repulsive forces act on it [90]. Obstacles generate repulsive forces while a goal attracts the robot. As in the case of the electric field the forces are stronger closed to the obstacles and to the goal so that far away objects have less effect on the robot movement [89]. The superposition of these forces generates a vector of desired robot movements.

In the basic approach the search space is decomposed into cells for which two forces $\vec{F}_{attr}$ and $\vec{F}_{rep}$ are calculated as follows:

$$|\vec{F}_{attr}| = -k_{attr}(x - x_{goal}) \quad (24)$$

$$|\vec{F}_{rep}| = \begin{cases} k_{rep} \left(\frac{1}{\rho} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2} \frac{\partial \rho}{\partial x} & \text{for } \rho \leq \rho_0 \\ 0 & \text{for } \rho > \rho_0 \end{cases} \quad (25)$$

where:

- k_{attr}, k_{rep} – scaling factors,
- ρ – a distance to obstacle,
- ρ_0 – a distance of influence,
- x, x_{goal} – a position of the robot and goal.

The direction of $\vec{F}_{rep}$ is perpendicular to the source of repulsive force while $\vec{F}_{attr}$ is in line with the segment connecting the current position and the goal point (see Fig. 13b). A potential field grid (Fig. 13a) constitutes a graph with $\vec{F}_{attr} + \vec{F}_{rep}$ for which a path is a set of consecutives minima. Steps of creating $\vec{F}_{attr}$ and $\vec{F}_{rep}$ are shown in Fig. 14.

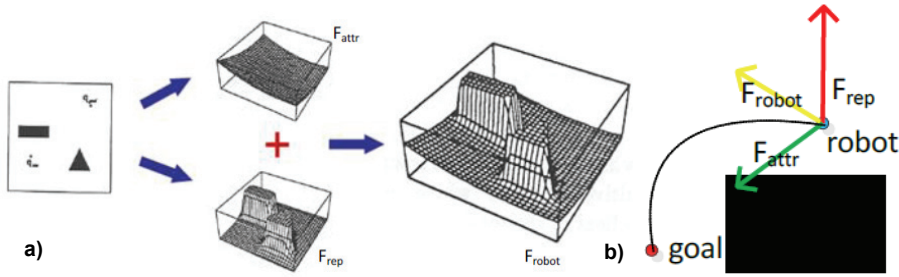


Fig. 13. An example of $\vec{F}_{attr}$ (green) and $\vec{F}_{rep}$ (red) and the resultant $\vec{F}_{robot}$ (yellow).

The main drawback of the potential field method is the presence of local minima that causes the robot to be trapped in [91–93]. An example of such a minimum is shown in Fig. 14d.

In order to avoid and overcome the local minima problem two techniques can be employed: (1) to escape the minima, and (2) to build a potential field without local minima [5]. The work of Kodzitschek [30] shows that every obstacle produces a saddle point in the potential field that lead to the occurrence of local minima.

The first group of methods consists in creating navigation functions. The simplest approach is to add some noise to the global navigation function [93, 94]. Connel, Rizzi and Choset [95, 96] proposed the creation of local potential fields, where locality is defined by a map decomposition. An additional approach for escaping local minima is a random walk based technique that in case of the robot being trapped in a minimum that is not a goal point, imposes a randomized force on the robot knocking it out of that minimum [97].

The artificial potential fields were extended by using the depth-first search, where a path was composed of segments of the same non-unit lengths towards the direction of a potential field gradient [33]. Choosing a small length of the step, a local minimum area can be avoided but longer steps narrow the size of the potential C-spaces because of the existence of obstacles. To overcome the local minima problem one can incorporate the best-first planning (search) methods [33].

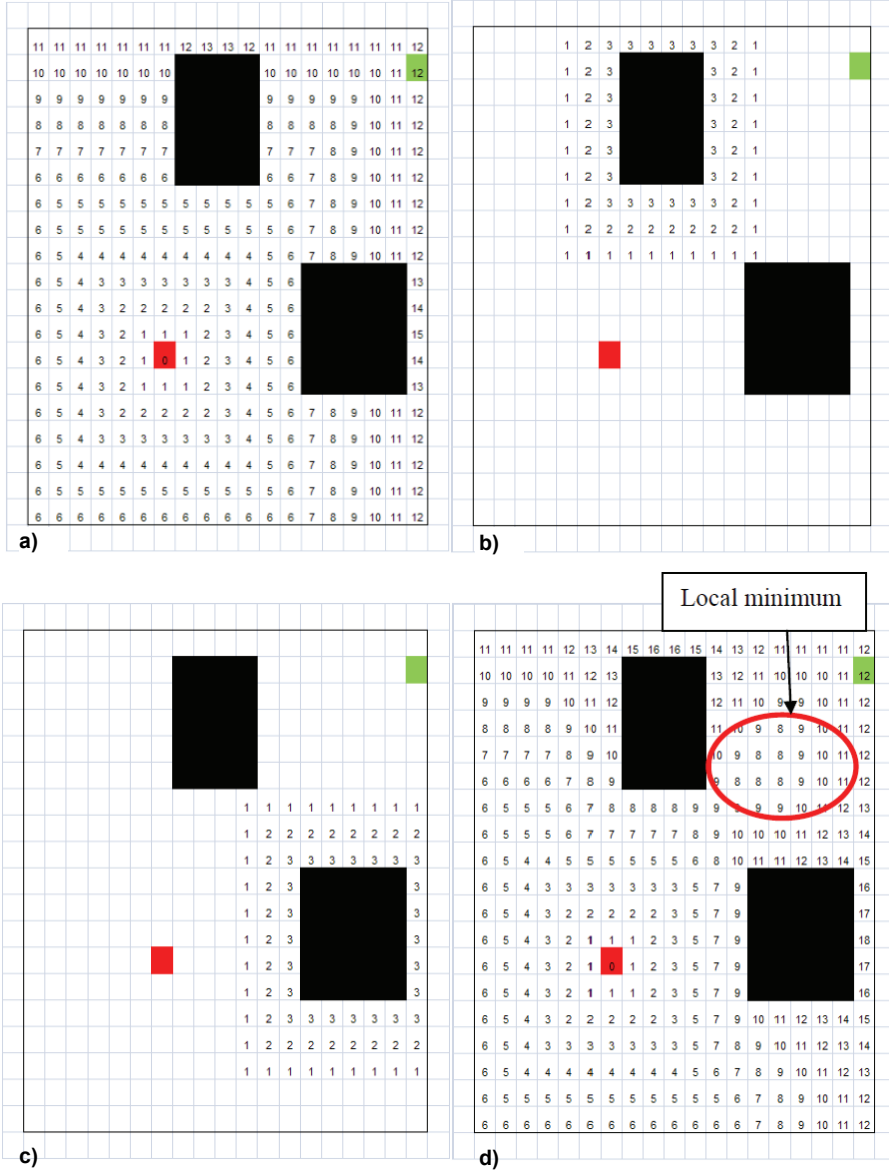


Fig. 14. An example of a local minimum for a regular grid artificial potential field.

a) $\vec{F}_{attr}$ calculated, b) and c) $\vec{F}_{rep}$ for both obstacles, d) a local minimum that would cause robot trapping.

The configurations in the C-space are used to construct a tree. In each iteration a branch with the lowest potential is chosen. The neighbors of such a cell are selected and evaluated by the value of potential while a pointer to these

neighbors is set to this cell. In this way a path through the graph containing a local minimum is terminated while another one is being searched through. If another path also contains a local minimum, then yet another one is been searched for and local minima in every step are filled up [69, 98].

In fact, the artificial potential fields methods were used in the past as fast planners, being not difficult to implement. It is relatively easy to create the potential field even for a point robot but there is a great difficulty for constructing the field for a higher dimensional C-space [97]. It is also difficult to take into an account different properties of obstacles or terrain types as it has to be included in the potential building function.

3.2.2. Wavefront planning

Similar results for path planning can be obtained using *wavefront planners*. The general idea of these algorithms consist in considering the C-space as a heat conductive material with heat radiating from the start node to destination node. An advantage of this approach is its completeness since the heat will eventually reach the goal configuration. The path produced is also optimal in the sense of length but it is difficult to incorporate other planning performance criteria.

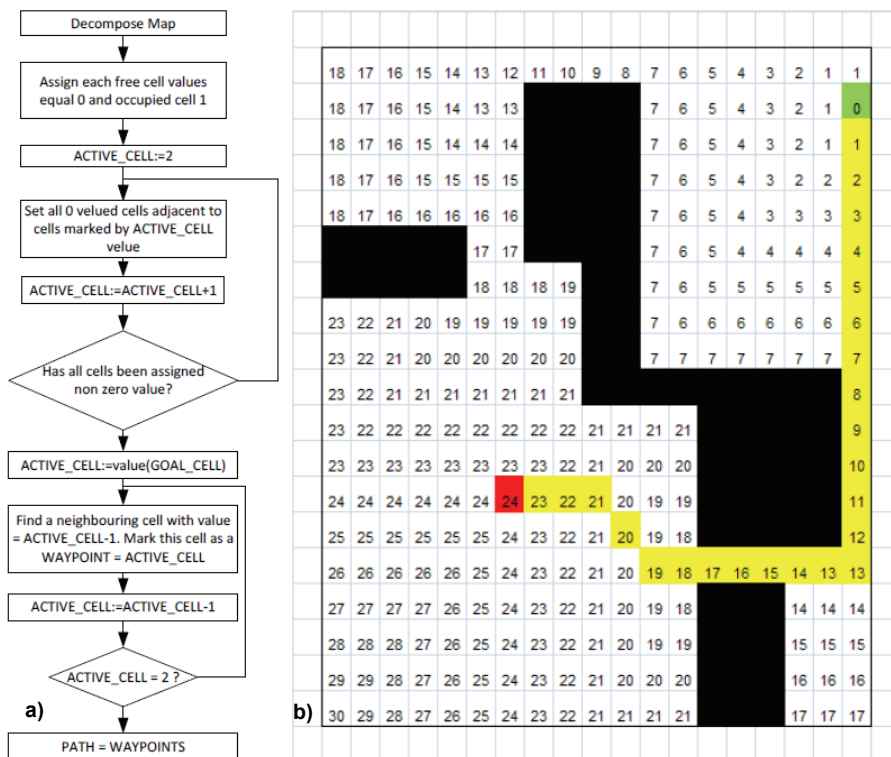


Fig. 15. The wavefront planners. a) Overview of the algorithm, b) An example of path planned by a wavefront planner (yellow). The numbers represent the total cost to reach a particular cell.

However, the wavefront planners can handle different terrains [99]. It is done by decreasing the speed of wavefront propagation through the particular regions having a non-zero traversal cost. Obstacles have a zero wave conductivity, open space has an infinite one, and undesirable traversable terrains have a low conductivity exhibiting an effect of high-cost paths [100].

The wavefront planner need a prior cell decomposition but in most cases an approximate cell decomposition is suitable for the planning purpose. An example of a path planned using a wavefront planner is shown in Fig. 15 with an outline and description of the algorithm.

Each cell after the decomposition is assigned a number corresponding to the cost of getting to that cell and calculated by the wavefront planner from the initial cell to the goal cell. After the costs are assigned to all cells, the points (cells) along the path are collected backwards by searching the neighbors with a lower cost (see Fig. 15a).

Some examples of the wavefront planners (Fig. 15) are the *NF1* and *NF2* algorithms by Latombe and Barraquand [11, 33]. *NF1*, implemented by Lengyel [101], and Brock [102] has a drawback that it directs the robot to go just by the obstacle border which is not desirable in sense of a safe robot movement, and it needs to use the LVP method for obstacle avoidance. *NF2* improves the *NF1* produced paths by maximizing the distance to obstacles and produces paths similar to the Voronoi diagram. Other methods based on more sophisticated graph searching techniques can be found in [103, 104].

3.3. Graph search techniques used in the GPP

Once the graph for traversing the map is created, a graph search technique has to be executed to find an optimal path. Graph based techniques use either cell decomposition methods to position a vertex position in the C-space or any randomized method for such a purpose. This chapter focuses on traversing the graph constructed rather than on graph building.

3.3.1. Depth First Search (DFS)

The *depth first search* is a general procedure for searching for a path within a graph and a tree. For a tree structure an algorithm searches each branch as far as the end of the branch is reached and then it backtracks to the vertex where the last fork point with a not-yet-visited branch was left. The order in which a graph or tree is searched is showed in Fig. 16.

Finding a path in the worst case is of complexity $O(|V| + |E|)$ ($|V|$ is the number of vertices, $|E|$ is the number of edges), i.e. the time complexity is linear. Each branch is searched through until a dead-end or until some predefined depth has been reached.

The output of the algorithm is a spanning tree of vertices (Fig. 16) of a finite depth. An iterative deepening depth-first search is used in the case of a depth limited search in which the depth-first search with a particular depth setup is run iteratively with an increasing depth. Therefore, deeper nodes are never explored until all shallower vertices are evaluated. The solution is then found in the n -th iteration of the depth-first search for a goal point placed on the n -th level. The time complexity

of this approach equals $O(d^b)$ (d is the depth of the goal and b is the average branching factor for all vertices that lie higher than d).

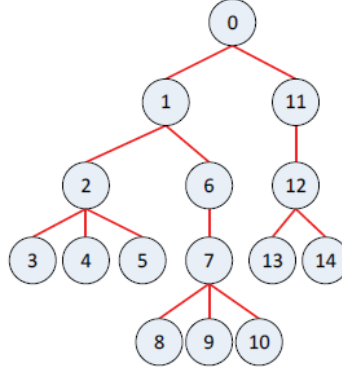


Fig. 16. The order in which nodes are searched in the depth-first search procedure.

The DFS algorithm is complete which means it always finds a solution if one exists regardless to graph type if the branching factor is finite.

3.3.2. Breadth First Search (BFS)

The breadth-first algorithm searches each depth level in one step, i.e. all vertices are explored on a particular level and then a next level is explored until a goal vertex is found (Fig. 17a). The time complexity is now as for the DFS, i.e. $O(|V| + |E|) = O(d^b)$ ($|V|$ is the number of vertices, $|E|$ number of edges) and is linear.

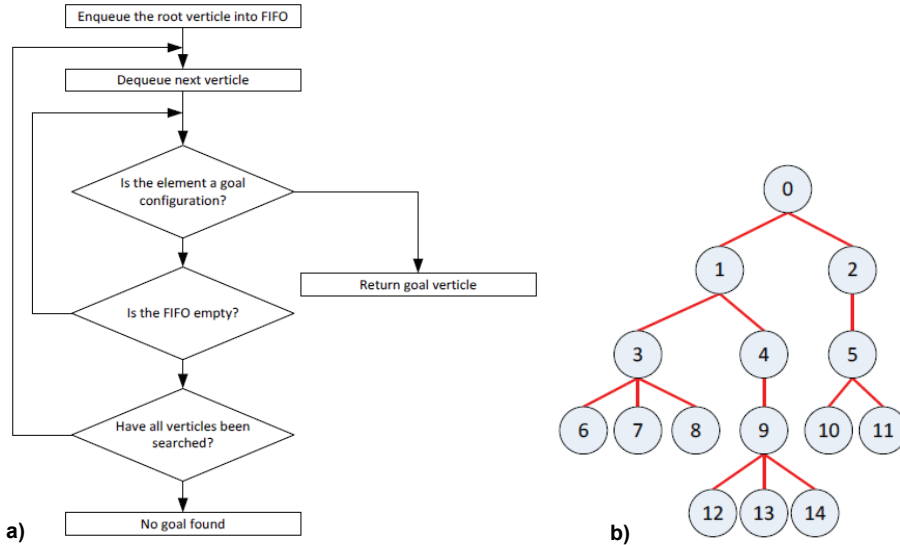


Fig. 17. a) Outline of the BFS algorithm; b) Order in which the nodes are searched in the BFS algorithm.

The BFS algorithm is complete which means that it always finds a solution if one exists regardless of the graph type if the branching factor is finite. As in the case of the DFS for a graph of a unit traversal cost, the BFS is optimal as it returns the path containing the smallest number of edges. For a weighted graph the method has to be extended to a uniform cost search that considers the traversal cost.

3.3.3. Uniform Cost Search (UCS) and the Dijkstra Algorithm

The *USC* is a weighted tree search algorithm that starts from the root vertex and searches the nodes in the order of increasing total traversal costs. All vertices are put in a priority queue in which the vertices are associated with the total traversal cost from the root (Fig. 18a). The node that starts the queue is expanded and its subnodes fill up the queue (Fig. 18b). Similarly as for the *DFS* and *BFS*, the *UCS* works for graph and trees with non-negative weights.

The algorithm is complete and optimal in the sense of being able to find the shortest path because all vertices are searched for the best transitional cost, and each time such a cost is found this more suitable way is updated to a new value [105].

The time complexity of the algorithm is $\tilde{O}(b^{C/M})$ where b is the average branching factor, C is the solution cost and M is the minimum edge cost.

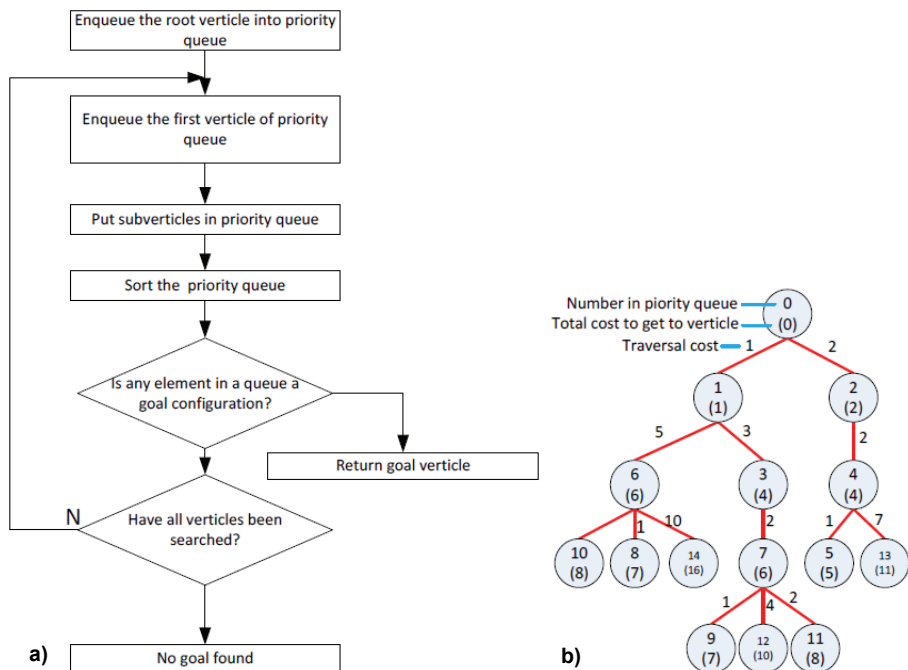


Fig. 18. a) An outline of the UCS algorithm, b) An example of a tree with the traversal cost and positions of vertices in the priority queue.

The Dijkstra algorithm [50] is a specific case of the *UCS* algorithm, in which a path from each vertex towards the start point is found. Therefore, all graph vertices

have to be checked and for any particular initial point an optimal path for all vertices are found. The algorithm was proposed in 1956. Its regular implementation runs in $O(|V|^2)$, but the use of the minimal priority queue based on the Fibonacci heap implies the time complexity of $O(|E| + |V|\log|V|)$ and is the fastest single-source shortest-path algorithm for a graph with non-negative weights [106]. The well-known Dijkstra algorithm can be used for finding multiple paths via a graph by iteratively finding an optimal path and removing one of the nodes that this path includes and searching the graph once again.

All algorithms discussed so far are plague by severe memory requirements. For instance in order to solve the graph search problem using the Dijkstra algorithm for 15 000 vertices one needs to construct $3 \cdot 15\,000^2 \cdot 4\text{ B} \approx 2.5\text{ GB}$ of tables that take linear space in computer memory while most of the 32-bit Operating systems can address up to 2 GB of memory.

3.3.4. Trulla algorithm

The Trulla algorithm is a graph search algorithm that is similar to the wavefront planning type *NF1* and *NF2* methods [11]. It was first introduced in 1991 by Mitchell [104] but the first implementation was by Murphy [103, 107]. It uses a decomposed map in which each cell not only contains information whether an obstacle is present there but also a (measure of) difficulty to traverse through a particular region as some paths may be more preferable than others.

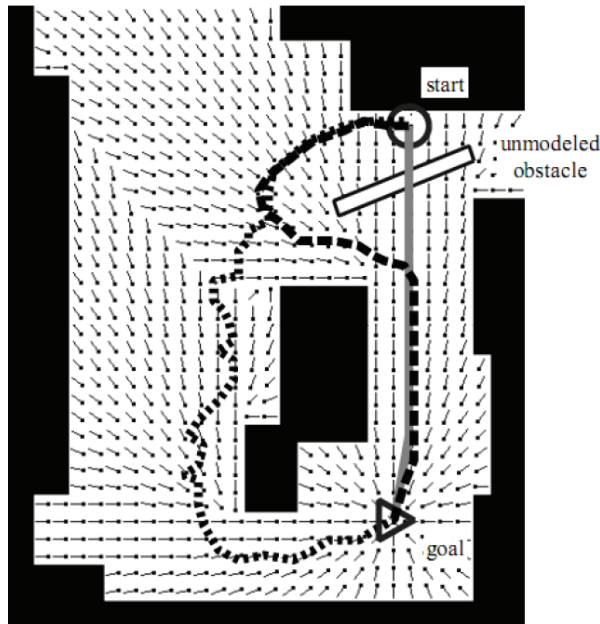


Fig. 19. An example of the Trulla *GPP* with a reactive *LMP*. The gray line shows the original path computed by using the Trulla algorithm (the rectangle obstacle was not modeled on the map), the log dashed line is the actual path obtained using the with Trulla while the short dashed line is a path obtained by the pure *LMP*.

Information that each cell contains is a vector that points to the goal and the magnitude value of which is proportional to the distance to the obstacle [108]. This feature means that an optimal path is found from each initial point of the cell and if an *LMP* (Local Movement Planner) calculates a subpath that deviates from that originally calculated, then the map can be easily updated. Therefore, the combination of the *GPP* and *LMP* is facilitated because a replanting procedure does not have to change all vectors in all cells so that a so called “halt-and-replant” behavior is avoided [108].

An example of the Trulla *GPP* and *LMP* reactive planner with an unmodeled obstacle is shown in Fig. 19. The optimality of a path is a function of the decomposition grid – the smaller the decomposition grid the “more optimal” the path is.

3.3.5. A* algorithm

The *A** algorithm is an extension of the Dijkstra algorithm that attains a faster running time by employing a heuristic approach. It was introduced in 1968 by Hart, Nilsson and Raphael [109]. Basically, the *A** algorithm uses the breadth first search [105, 110] but instead of a purely deterministic cost function it prioritize the search queue (OPEN_LIST in Fig. 20a) by a heuristic distance-plus-cost function:

$$f(c) = g(c) + h(c) \quad (26)$$

- $g(c)$ – a cost to get to configuration c ,
- $h(c)$ – an admissible heuristics estimate of a distance to the goal configuration.

The function $h(c)$ has a important property of not overestimating the distance of a current configuration to a goal configuration, and for any vertex pair c_1, c_2 , $h(c_1) \leq d(c_1, c_2) + h(c_2)$, i.e. h has to be consistent. If has such properties, then no configuration is estimated more than once.

An extension of $f(c)$ by $h(c)$ sorts a priority queue in the way that directions towards the goal are preferable, and therefore proper metrics have to be used because an overestimation of this distance leads to non-optimal paths.

The $g(c)$ corresponds to a cost function to get to node c and the *A** algorithm may use it as a path cost. By doing so, a proper $h(c)$ has to be taken into account as simple metrics could direct the search into regions of a higher cost and produce non-optimal paths. A variant of *A** called a weighted *A** can put more emphasis on heuristics by amplifying their heuristics:

$$f(c) = g(c) + \alpha h(c), \quad \alpha > 1 \quad (27)$$

This should be done if and only if $h(c)$ is admissible because if a non-deterministic heuristic function leads to a non-optimal path, then this effect this effect will be amplified. When $\alpha > 1$, the search process is sped up, but the path produced is sub-optimal and the total cost of such a path can be α times worse than for $\alpha = 1$.

If $h(c)$ equals zero, then the *A** algorithm becomes the breadth first search. Its time complexity depends on the time complexity of $h(c)$. In the worst case it is exponential with respect to the length of solution. However, as Pearl [110] observed

and Russell and Norvig [105] showed, for a heuristic that is close to an optimal one, it is $h^*(c)$ ⁵ so that:

$$|h(c) - h^*(c)| = O(\log h^*(c)) \quad (28)$$

However, still a literature review shows that A^* can be used as a real-time algorithm [111]. In Fig. 20 two examples of paths obtained by using the A^* method are presented for regular heuristics $h(c)$ (Fig. 20b) and for $\alpha = 5$ (Fig. 20c) [112].

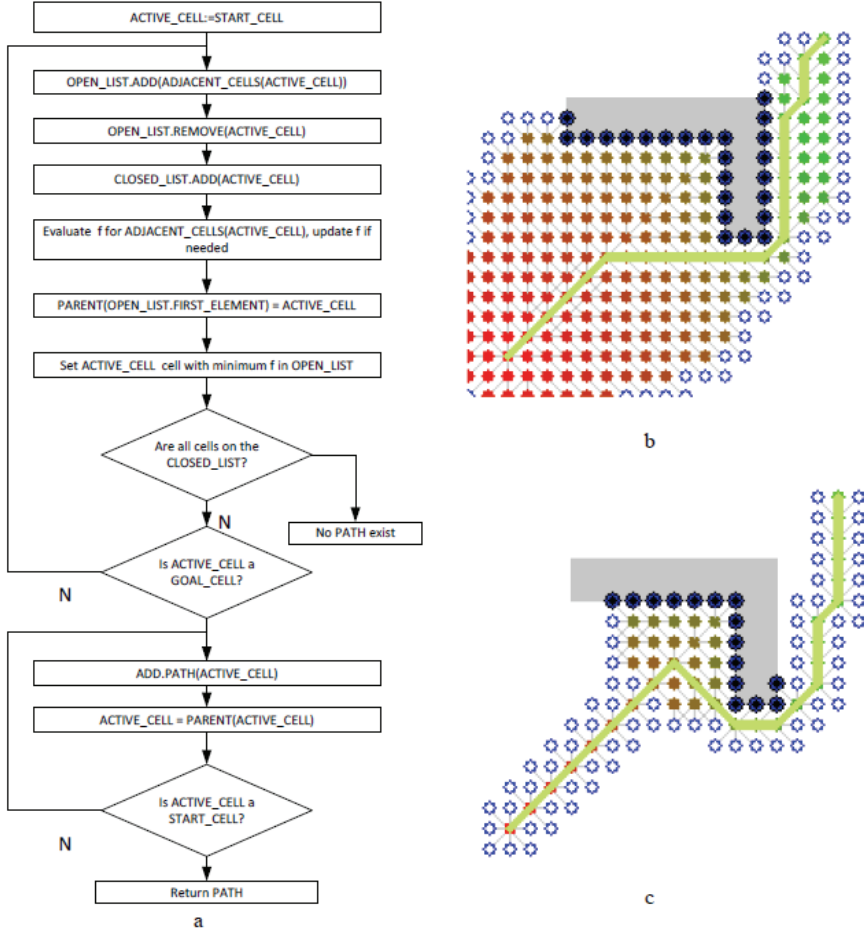


Fig. 20. Illustration of the A^* search for finding a path from a start vertex to a goal vertex. The empty circles represent the nodes in the open set, and the filled red/green ones are in the closed set. The green line represents a path. a) Outline of the algorithm, b) $\alpha = 1$, c) $\alpha = 5$. Cf. Bhattacharya.

⁵ That is the true distance.

There are many variants of the A^* algorithm, and the most important of them are listed below:

- Incremental search techniques. These methods include D^* [113], Focused D^* [114], Field D^* [115], D^* Lite [115], IDA^* [116], Fringe Saving A^* (FSA^*) [117], Generalized Adaptive A^* (GAA^*) [117], Lifelong Planning A^* (LPA^*) [118]. Three main groups of incremental search techniques [119] can be distinguished. The first class which restarts the search technique, i.e. updates the heuristics and cost functions, the second class which updates the heuristic function value, and the third class which updates the cost function value. The family of the D^* algorithms are used for planning a path with a partially unknown terrain but an algorithm has to know which part of a map is unknown. It is based on an assumption that an unknown terrain has a unique feature, i.e. it belongs to C_0 or C_{free} . These methods are *SLAM* (Simultaneous Localization and Mapping) ready because if any change on the map is not up-to-date, e.g. some obstacles exist while they are not presented on the map, they are added and a new shortest path is calculated. Replanning has to be done in real time so that the time complexity is an issue in the D^* algorithms. It uses incremental heuristic search algorithms [118, 120, 121]. Additionally to `OPEN_LIST` and `CLOSED_LIST` (see Fig. 20a) the updating of the node cost needs to annotate vertices as `RAISE` for vertices having a higher cost than before the update on the `OPEN_LIST`, and `LOWER` for those where the cost decreased from the last update and which are on `OPEN_LIST`. The Pure D^* algorithm is not used in practical implementations because of a complicated implementation. Instead, the D^* Lite algorithm by Koenig and Likhachev [122] is employed that is an incremental heuristic which combines A^* with the SWSF-FP algorithm.
- The IDA^* algorithm is a version of the A^* algorithm that lowers the space complexity of the algorithm. It uses the same informed search method of $g(c)$ as A^* , but the cost and the shortest path is remembered only for the first element of `OPEN_LIST` and not for all nodes visited.
- The FSA^* algorithm is a first class incremental search technique. It is the simplest approach in which the A^* algorithm is restarted from the point where the path deviates from the one previously calculated.
- In the GAA^* approach a heuristic function value $h(c)$ is updated in the next search to make $f(c)$ more precise.
- The Θ^* algorithm [123] is an extension of A^* in which the paths found by the algorithm are more natural by introducing a so called any-angle planning. It is done by replacing `PARENT(OPEN_LIST.FIRST_ELEMENT) = ACTIVE_CELL` (Fig. 20a) block by a decision block shown in Fig. 21a. This combines the visibility diagram method with A^* by inheriting the parent from the parent of an active cell if such a visibility occurs. This increases the time complexity of A^* by the visibility calculation but produces more smooth paths. An example of a path derived by using Θ^* is shown in Fig. 21b. If the decomposition reaches the bottom granularity, the paths produced by Θ^* and A^* are identical but in general the numbers of subgoals are smaller for Θ^* .

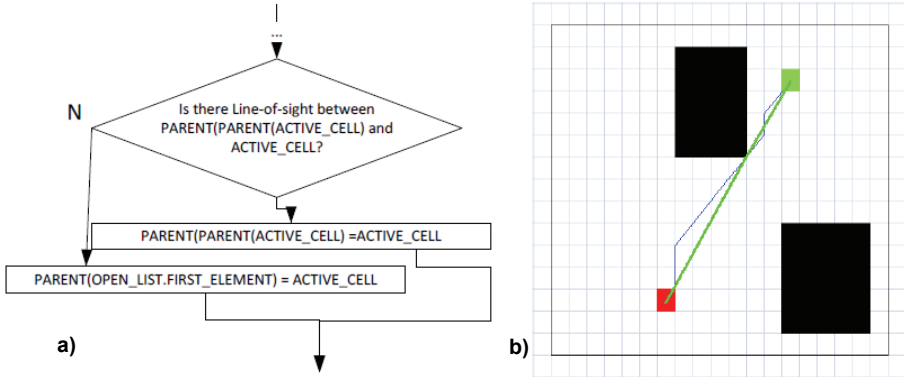


Fig. 21. a) A modification of the Theta* over A* by parent inheritance – cf. the block $\text{PARENT}(\text{OPEN_LIST.FIRST_ELEMENT}) = \text{ACTIVE_CELL}$ of Fig. 20a. b) An example of the Theta* (green line) and A* (blue line).

- The *SMA** (Simplified Memory-Bounded Algorithm) [124] is an extension of the *A** algorithm that uses for computing a path the available memory only. Therefore one can limit the OPEN_LIST length a priori before running the algorithm. The block $\text{OPEN_LIST.ADD}(\text{ADJACENT_CELLS}(\text{ACTIVE_CELL}))$ (Fig. 20a) is replaced by the block shown in Fig. 22. Note that as the element from OPEN_LIST is removed, its parent has also to be removed from CLOSED_LIST. If the vertex has been in CLOSED_LIST simultaneously when any parent of that parent is on the CLOSED_LIST, then it also has to be removed. This leads to an increase of the running time as one vertex can be added to OPEN_LIST and removed from CLOSED_LIST many times [124].

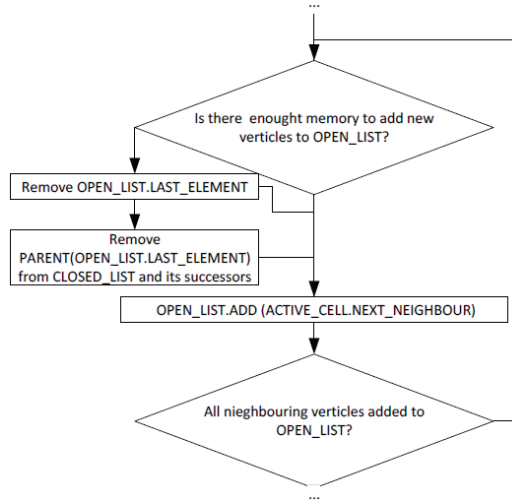


Fig. 22. The block replacing $\text{OPEN_LIST.ADD}(\text{ADJACENT_CELLS}(\text{ACTIVE_CELL}))$ (Fig. 20a) in the *SMA** method.

4. Advanced heuristic methods in the GPP

The GPP problem is known to robotics society for over 40 years and still new methods are being developed to search for a path. This is because many well-known methods provide optimal solutions but are very sensitive to the formulation of the optimization problem. Changes of goals require the use of much more complicated conceptually and numerically multi-objective optimization techniques should include different and varying constraints on path planning and on computational resources available for the task, different environments, etc. For these reasons new advanced ant-colony, genetic algorithms and artificial neural network meta-heuristic based methods have been introduced to overcome those inherent difficulties [125, 126].

4.1. Evolutionary computing in the GPP

4.1.1. Genetic algorithms in the GPP

The problem of offline planning for the case of a well-known environment can be solved by algorithms mentioned previously [113–117, 127] and many similar methods, but still they are inflexible and sensitive as mentioned above. They solve the problem as it was a single criterion optimization one, and the performance function considers in most cases the path length.

Evolutionary computation is in general a powerful tool for solving a wide variety of optimization problems including the *NP*-hard ones like the *SAT* problems, *COP* (constrained optimization problems), dynamic optimization problems, etc. [105, 128–130]. In this chapter most viable significant elements of Genetic Algorithms for solving GPP will be presented as well as more examples of path produced by selected methods.

The first approaches to use genetic algorithms for solving the GPP problem appeared in the early 1990s [131–133] but presented problem formulation was simple, without incorporating any problem specific knowledge, and therefore a convergence was poor and much computation time had to be spend on path optimization. Although genetic algorithms have been subject of rapid development in both: sophistication level of used methods, mechanisms or strategies and a field of use there are several aspects inseparable from using these. Amongst these the most important are:

1. Evolutionary Computation methods belong to metaheuristic group. This means they are good solvers for a variety of search, optimization and machine learning tasks but on the other hand they do not guarantee finding a “proper solution”. Moreover they do not guarantee finding any solution at all, but in practice the quality of results is highly comparable with classical methods.
2. Multiple launches of Evolutionary Computation based algorithms naturally ends-up with slightly different results, therefore comparing results from a single launch is a proper approach only for presentation purposes. Only

a stochastic analysis of results from multiple runs is a proper approach for result comparison.

3. For Constrained Optimization Problems such as Global Path Planning randomly generated candidate solutions are chiefly infeasible. Simultaneously feasible solution may produce infeasible offspring. The elaboration of heuristic knowledge based genetic operators is at the very least difficult.

Later, newer genetic algorithm based planners [134, 135] included adaptive approaches by changing genetic algorithm parameters, operators and the fitness evaluation function [136, 137] or by including self-tuning properties [138]; conceptually, those algorithms looked very similar to the genetic algorithms following their general framework (cf. Fig. 23).

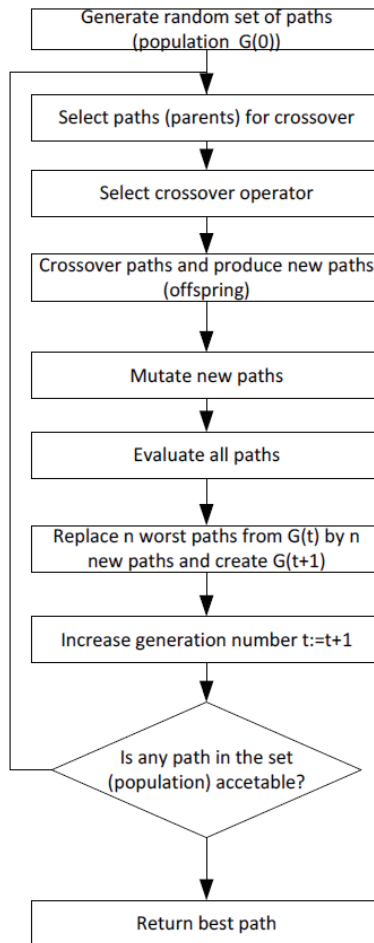


Fig. 23. A general framework for the GPP using a genetic algorithm.

The main differences between genetic algorithms used for solving the GPP problem are in the definition of operators of crossover and mutation, the evaluation function, the convergence conditions, the population model, etc.

Some algorithms [135] use an initial population calculated by other well-known path planning methods. These speeds up computations by feeding the algorithm with solutions that are initially at least feasible if not near-optimal. This method in the genetic algorithms is called seeding and, as shown by Michalewicz [129], it can lead to a pre-mature convergence to a local minimum, and should in general be avoided.

The essence of path planning by using the genetic algorithms lies in the chromosome representation. Some authors prefer sampling configurations from the C-space [138, 139], some simple horizontal/vertical vector steps [140]. Also random walks in the sense of angle-displacement [141] or vector or vertical-horizontal displacement or graph vertexes [142] are used. Both specified lengths [143] and variable length chromosome representations [135] are used but in most cases, for simplicity, a constant length of the genotype is used. Shidong [144] presents an approach when a particular allele are coded as a transitions in path network created by quad tree decomposition (see chapter 2.5), but it requires a fixed length coding to accomplish crossover and mutation, however it always finds feasible solutions. Nevertheless most of present evolutionary procedures for solving GPP use one, particular configuration of the robot i.e. one position to reach [131–133, 145, 146] to represent a particular allele. Such representation has several advantages over others. First of all it expels a possibility of movement outside the map; therefore no violation treatment concerning this issue is needed. Secondly when sampling for a configuration is performed it is fairly easy to determine whether the configuration is or not violating in sense of obstacle encroaches. Finally the quality of information that is transmitted over generations does not depend on information length as each configuration (point on the map) can exist separately to other points while still being valid. Apart from these advantages another important aspect is essential: a genotype has a form of a set rather than a permutation and an implementation of mutation and crossover operators is intuitive. However as in the case of every such type of representation one significant aspect of a path is not being maintained over generations for such representation: the shape of a resultant path. It is vital to understand that the path is as good as methods of particular obstacle omitting within that path are. Therefore another type of genotype representation is proposed. In the scope of this work a genotype A_s of a specimen s is represented as permutation of segments of movement.

None of the representations mentioned can guarantee that no obstacle violations occur and therefore a proper evaluation function is required. However several approaches have been made to overcome this problem. Yun, Ganapathy and Chong [145] incorporate two mechanisms to deal with this issue, namely an *Obstacle Avoidance Algorithm* and a *Distinguish Algorithm* (Fig. 24b). The *Obstacle Avoidance Algorithm* consists in sampling a configuration space in a specific manner. If a sampled configuration does not belong to an obstacle i.e. it is non-violating it is left as a possible to use allele, if not a new point in generated according to following procedure (Fig. 24b):

1. Find a distance $|AB|$ (A is a point that violates obstacle).
2. Draw a right triangle ($\triangle ABC$) and find an angle α from the equation
$$\alpha = \tan^{-1} \frac{y_A - y_B}{x_A - x_B}.$$
3. Find point C as an intersection of lines containing $\overline{BC}$ and $\overline{AC}$ ($\overline{AC}$ is perpendicular to $\overline{AB}$).
4. Replace point A with point C .
5. Generate next allele.

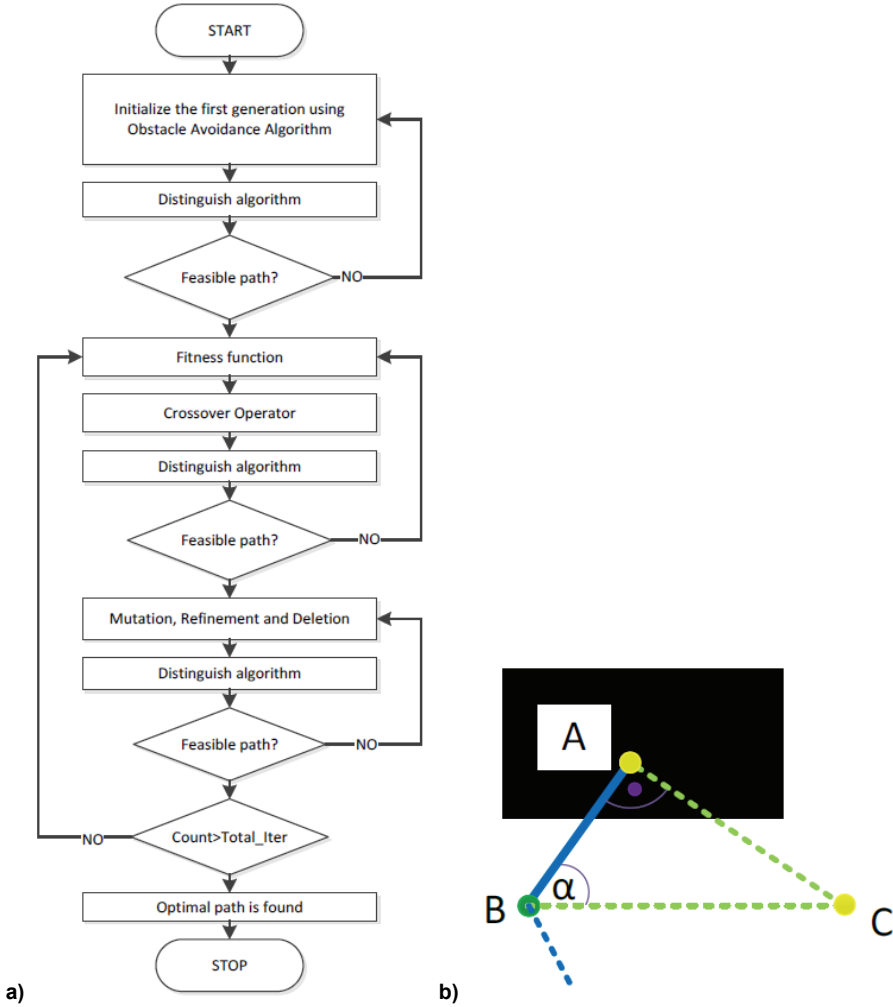


Fig. 24. a) An outline of an example of improved genetic algorithm [145], b) Distinguish algorithm as a modifying operator [145].

This ensures that particular points have their visibility and whole path does not violate obstacle encroachment planning rule. Additionally to this Yun proposes Distinguish Algorithm (*DA*). The underlining assumption is that *DA* aims in verifying if any of the points between two nodes m and n falls inside the obstacle area. If the coordinate of an intermediate point is placed with obstacle an algorithm checks such intersection for the next intermediate point. If none of the intermediary nodes lies in the obstacle area the path is categorized as feasible. For more detailed description of the procedure refer to [145].

In most cases paths derived by genetic algorithms are optimized taking into account several objectives from which the most common are [128, 131–133, 139–142, 147–149]: length of the paths, number of intermediary points, path smoothness, clearance from obstacles, etc. The necessity to account for multiple objectives is overcome by using a single evaluation function [128, 131–133, 139–141, 147–155] or, at most, a bi-criterion optimization is used [128, 129, 142].

On the other hand, numerous operators have been defined in the form of mutation [132, 135, 138]. The implementation of these operations differ from each other in particular because of different chromosome representation but basically they lead to similar results.

Among many example of such specific adopted solutions, one can mention:

- *Translational mutation* (Fig. 25a) – a particular configuration within the chromosome changes its coordinates but connections to and from it remain unchanged.
- *Insertion mutation* (Fig. 25b) – an additional configuration between two adjacent configurations is added.
- *Deletion mutation* (Fig. 25c) – a particular configuration is removed from the chromosome.
- *Swap mutation* (Fig. 25d) – the configuration permutation is changed by swapping two (in most cases adjacent) configurations.

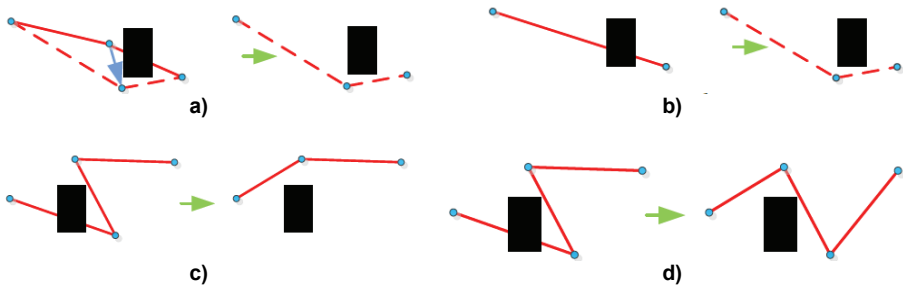


Fig. 25. Typical mutation operators for the GPP problem solved by using the genetic algorithms. [128, 132, 139–141, 147]. a) Translational mutation, b) Insertion mutation, c) Deletion mutation, d) Swap mutation.

The mutation operators can be improved and deteriorated and therefore in a more sophisticated approach proposed by Xiao [138] the mutation operators are characterized by their effectiveness, i.e. the ratio of the number of improvements

executed on the chromosome to the total number of operator executions and the average time of execution of a particular operator.

Newer approaches for solving a Global Path Planning Task aim in either use of specific genetic algorithm or use a specific support in a form of local search to improve random genetic search. Sadati and Taheri [156] propose to use the Hopfield Neural Network planner in conjunction with genetic algorithm. The Network is launched regularly to improve the quality of each chromosome by reforming it. They also propose a new method for seeding the initial population by regular sampling of the environment with following procedure:

1. Making grid of the environment by parallel lines both in X and Y directions and denote intersection points by $a_i, i = 1, \dots, n$.
2. Set k to 1.
3. Find a path from start point to a_k and from a_k to destination point.
4. Increase value of k .
5. Repeat steps 3 through 4 for each a_i .

An example of such specimen generated by this seeding mechanism is shown in figure below (Fig. 26) for $k = 3$ and for $k = 45$. Such seeding, however seems very attractive as it can be considered as best-effort method for sampling search space uniformly, has two disadvantages. First of all the number of unique specimen in initial population depends on grid size and equals up-most k^6 . What is more for different nodes a_i resulting path may be the same – the more cluttered the environment is the higher the possibility of such situation to occur is.

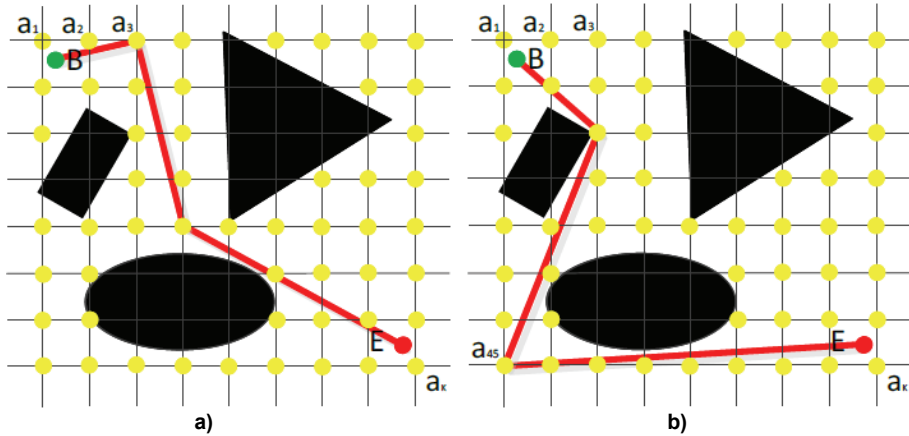


Fig. 26. An example of specimen in initial population generated by seeding method proposed in [156] a) for $k = 3$ b) for $k = 45$; legend: ● – path start point, ● – path destination point, ● – grid intersections (graph vertices), — path.

⁶ The number of unique specimen depends on map content as the grid intersection points can occur in occupied region.

This introduces a typical problem for seeding in genetic algorithms: initial loose of diversity. If a path which phenotype is likely to occur in several instances in initial population is in addition fitter than average it could start to dominate in a selection step (Fig. 24a) resulting in diversity drop of population.

An interesting suggestion on mutation operators have been made by Huahua and Zezhong [157]. They propose three original operators namely *Restoration Operation*, *Reconstruction Operation* and *Recording Operation*. These operators work by checking the value of the maximum⁷ fitness function in the current generation $f_m(t)$ and the upper bound of historical maximum fitness values $f_m = \max_i(f_m(t_i))$. The *Restoration Operation* rejects the current operations results (Mutation and Crossover) if $f_m(t) < f_m$ and restores the current population. *Reconstruction Operations* replaces current population with an initial population if mutation and crossover failed several times consecutively. *Recording Operation* saves the history of $f_m(t)$ and updates f_m . These operators work on a level of population and were elaborated to keep a constant progress of convergence within a population. Both *Restoration Operation*, *Reconstruction Operation* prevent from stagnation of population by consecutive search “restarts”, whilst *Recording Operation* plays a role of fitness historical database. Every time the algorithm is stuck with local minimum a *Restoration Operation* targets the current search into different direction on a fitness landscape and if this procedure does not yield in better population fitness whole population is being restarted to an initial state and search procedure is likely to find another solution.

The crossover operation in nearly all cases quoted in the literature is a one point crossover with no cross point selection and by selecting two parent paths producing one offspring path [135]. The crossover operator construction depends directly on chromosome representation. In a specific case of GPP the “ideal” crossover operator is able to handle the problem of flaw offspring. Such treatment is done in one of two possible ways: (1) parent selection mechanism is specialized in sense of selecting parent’s which offspring would not violate obstacles or (2) another improvement method is implemented to change the chromosome from violating to non-violating. The latter one can be considered as simultaneous crossover and mutation operator. Such operator should not work in an ideal manner: newly created offspring need to be in some cases fitter and in some worse as this is a diversity preservation mechanism.

An interesting method of offspring’s chromosome improvement is presented in [156]. It bases on the approach presented in [158] and it is called “Correction and Optimization” procedure. First an offspring created by one point crossover is performed on two parents (Fig. 27a and Fig. 27b) to produce an offspring (Fig. 27c). If there are alleles of non-visibility (Fig. 27c, green line) a Hopfield Neural Network is launched to inject additional alleles into chromosome and create new, non-violating specimen (Fig. 27d).

⁷ for maximization problem,

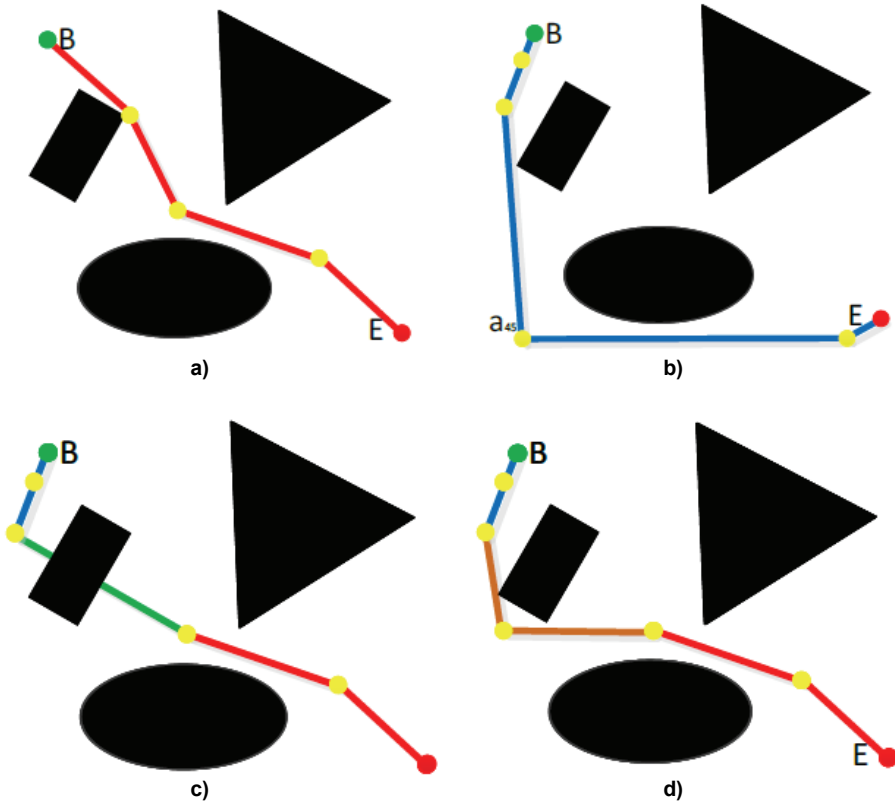


Fig. 27. An example of specimen in initial population generated by seeding method proposed in [156] a) for $k = 3$, b) for $k = 45$; legend: ● – path start point, ● – path destination point, ● – grid intersections (graph vertices), — 1st path, — 2nd path, — violating connection, — new injected segments replacing violated connection.

Another type of genetic algorithm to solve GPP task is a Quantum Genetic Algorithm (QGA) introduced by Yong et. al. [159] and then improved by Chuanling [160]. QGA is a combination between GA and quantum computing. The chromosome representation bases on quantum vectors – qubit coding, whilst the chromosome is updated by quantum rotation gate. As opposed to standard binary coding the Single portion of information in QGA is not coded as ‘0’ or ‘1’ but rather as a superposition of two at the same time [161] which is often written as:

$$\alpha|_0 + \beta|_1 \quad (29)$$

α, β – probability amplitudes associated with 0 state and 1 state. α^2 and β^2 represent probability of seeding 0 and 1 and $\alpha^2 + \beta^2 = 1$.

For GPP purpose a genotype is therefore coded as:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_s \\ \beta_1 & \beta_2 & \dots & \beta_s \end{bmatrix}, \alpha_i^2 + \beta_i^2 = 1 \quad (30)$$

Mutation is realized by an operator called Quantum Gate Optimization and is as in the case of regular Genetic Algorithm a transition of allele in the genotype. It operates as follows:

$$U(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (31)$$

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = U(\phi_i) \circ \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix} \quad (32)$$

- $\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix}$ – a new allele of a genotype,
- $U(\phi)$ – quantum rotation gate,
- $\phi_i = s(\alpha_i, \beta_i) \Delta \phi_i$ – sign of rotation angle,
- $\Delta \phi_i$ – direction of rotation, set between $0.005 \sim 0.1\pi$.

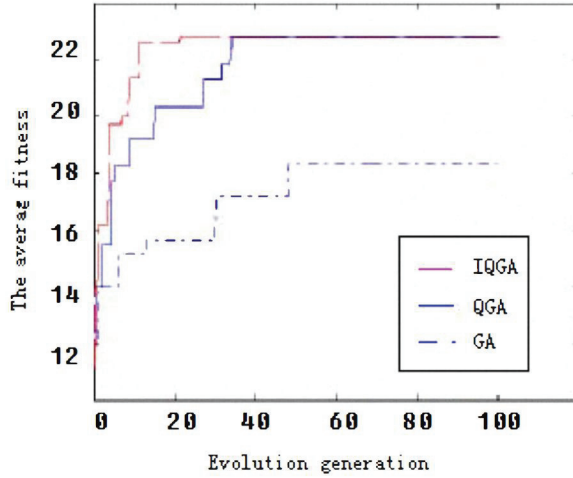


Fig. 28. A comparison of the performance of Genetic Algorithm (GA), Quantum Genetic Algorithm (QGA) and Improved Quantum Genetic Algorithm (IQGA). After Liu Chuan-ling [162].

Therefore quantum rotation gate is a mutation operator, which range depends on $\Delta \phi_i$ value. Since all allele's values come from stochastic therefore using Quantum Genetic algorithm one can expect a phenomenon of pushing all operators towards randomized search. Therefore an emphasize is put on a task of exploration

of a search space. On the other hand a convergence in a sense of speed will be worse than in classic genetic algorithm. A comparison of average fitness in function of generation for classical GA, QGA and Improved QGA presented in [161] is presented in figure below (Fig. 28).

Another type of genetic algorithm employed to solve GPP is a Hierarchical Genetic Algorithm (HGA). There are several evidences of a good quality search by this approach, one of most famous is a design of digital IIR filter [163]. The underlying idea for HGA is to partition a problem into a set of few smaller issues. In [164] authors propose to firstly (1) find a sequence of obstacles to omit and afterwards (2) to determine a relationship between particular obstacles and a robot. The first problem is similar to travelling salesman problem (TSP) [165–167] whilst the second can be solved with Simple Genetic Algorithm with a binary coding.

Some examples of paths derived by using the genetic algorithms by different authors are presented in Fig. 29.

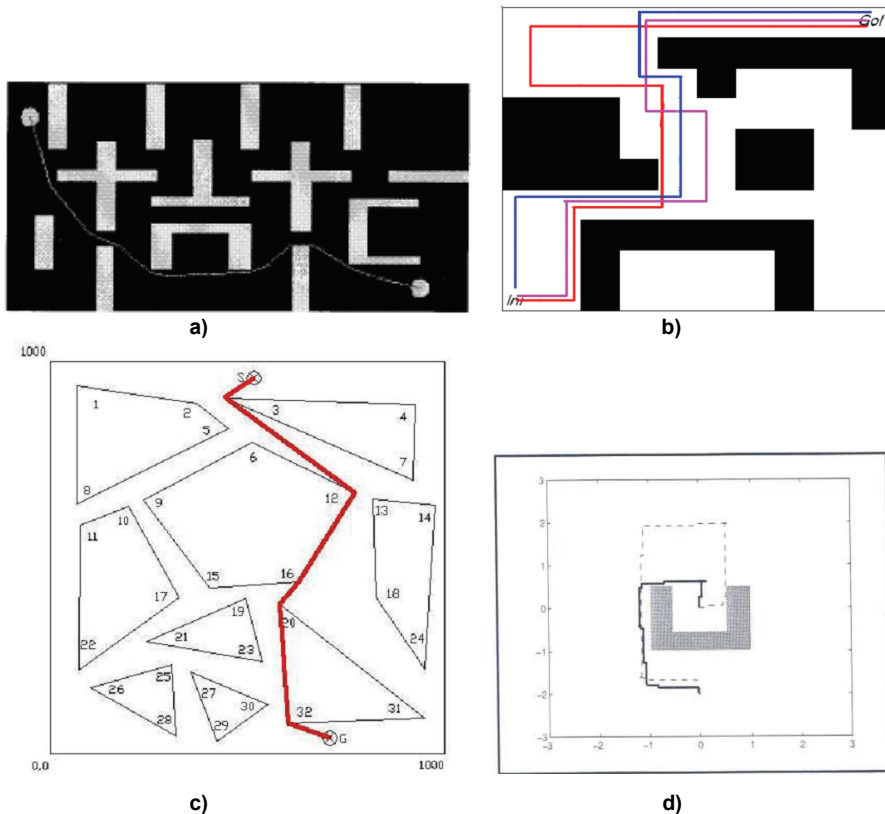


Fig. 29. Examples of paths planned using the genetic algorithms: a) an adaptive evolutionary planner/navigator for mobile robots [135], b) a static environment-based path planning method by using the genetic algorithm [168], c) mobile robot path planning using a hybrid genetic algorithm and the reversibility vectors method [149], d) a memetic algorithm based path planning for a mobile robot [143].

In all cases it is difficult to compare results obtained by different authors. This is because the basic framework of a genetic algorithm is so openly viewed and different authors introduce different approaches without giving more specific measures of performance – except for Xiao [135, 136]. In general, the methods presented still suffer from inefficiency by not introducing a path cost and advanced genotype repair functions. And even more, the examples presented in the literature are simple binary maps and the robot is treated as a point reducing the number of optimization and punishment criteria. Most of adaptive algorithms have also poor convergence, cf. for instance Fig. 29a, which shows a path generated in iteration number 800.

4.1.2. Memetic algorithms for the GPP

A *memetic algorithm*, similarly as each evolutionary approach based algorithm, is a population based search technique with all merits of the recombination operator. As opposed to the traditional evolutionary approach, the memetic algorithms concentrate on a maximal utilization of all available data aimed at obtaining an increase in search speed. The memetic algorithms, which have recently enjoyed an increased popularity [41], improve the quality of the search process since, as it was shown by Hart and Belew [152] or Wolpert and Macready [169], the quality of the search algorithm mirrors the quality and quantity of a priori knowledge that it uses. This thesis is well-known as the “No free lunch” theorem [97, 169–172].

In general, the improvement over the genetic algorithms consists in using other heuristic methods, approximation, local search techniques, specialized crossover and mutation operators, etc. [173], that is, advanced and problem specific methods of maintaining population diversity, heuristic seeding of initial population, etc. The success of memetic algorithms can be explained as a direct consequence of a synergy of different methods they employ. The memetic algorithms are therefore often called *hybrid genetic algorithms* which is often misleading as there is a slight difference between these.

The idea of a memetic algorithm can be outlined as shown in Fig. 30. The initial population within the memetic algorithm can be initialized by replacing the standard random population generation. This can be done by *Seeding* in which the initial population comes from the specimens generated by other constructive heuristics. It can also take the form of a selective initialization [174] in which a large number of potential specimens is generated and the best ones are chosen by their rank, in μ tournaments or by generating a number of solutions, taking one and performing mass mutation to create a population around the best one obtained from random sampling. Radcliffe and Surry claim in their work [175] that by using a totally random initialization one ends up in a better average performance⁸ than by initializing the whole population using constructive heuristics. The use of a small proportion of the initialized population together with randomly generated specimen increases this performance even more.

⁸ Measured as the dynamics of average fitness over generations.

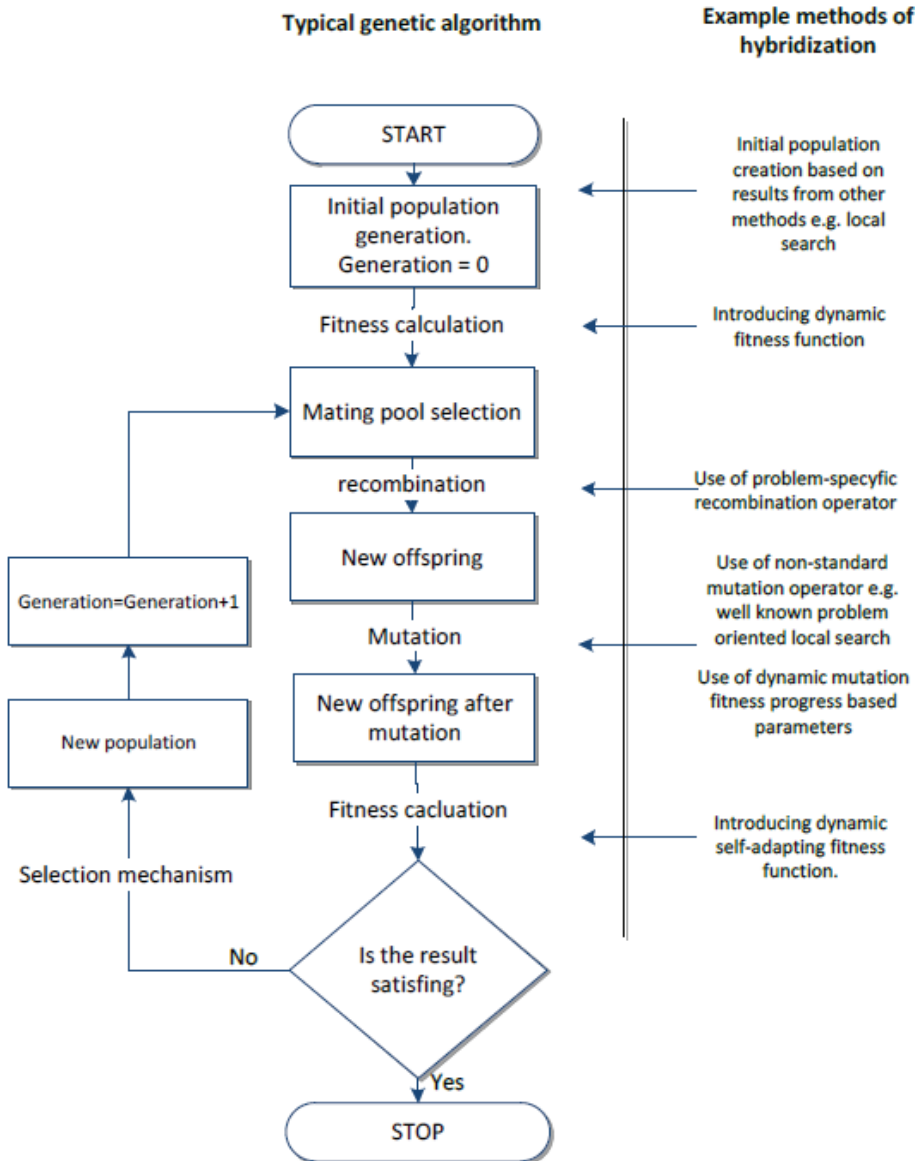


Fig. 30. Possible hybridizations of the genetic algorithm which lead to the memetic algorithm.

Another way to hybridize the genetic algorithms is to use intelligent recombination and mutation operators. The simplest way is to bias the mutation operator to change the genotype more often from 0 to 1 than vice versa. A simple recombination can be replaced by an operator that performs the recombination several times and the best option is chosen as an offspring. The most complex operators have two purposes: to produce a best possible offspring and to preserve

diversity in the population. An example is a Distance Preserving Operator for the travelling salesman problem (TSP) program by Merz [165]. Another way is to employ a local search after or instead of the mutation. This forms the idea that is the closest to the concept of a *meme* as an additional learning phase of the memetic algorithm [176–178]. Basically, as Krasnogor showed in his PhD dissertation [177], it is possible to introduce such a local searcher which shifts the genotype in not the same way as the mutation and recombination do since this limits the worst case run time of the memetic algorithms.

A perspective of the performance of the memetic algorithm compared with other methods including the pure evolutionary computing is presented in Fig. 31. The “quality” (Y-axis) is not defined in detail but the figure shows a general result after several years of research which clearly indicate that the memetic algorithm is a powerful tool for solving a wide variety of problems and by using appropriate, constructive heuristics in addition to the general evolutionary computing scheme we can expect an increased algorithm performance. What is even more: by selecting proper local search techniques we can shift the applicability of the method to a wider range if problems as sketched in Fig. 31. Therefore by choosing a proper hybridization method we can easily tune the genetic algorithm for the need of a specific problem, and this tuning can be done dynamically, i.e. by first finding a coarse solution using a genetic algorithm and then by amplifying local search techniques to obtain better results.

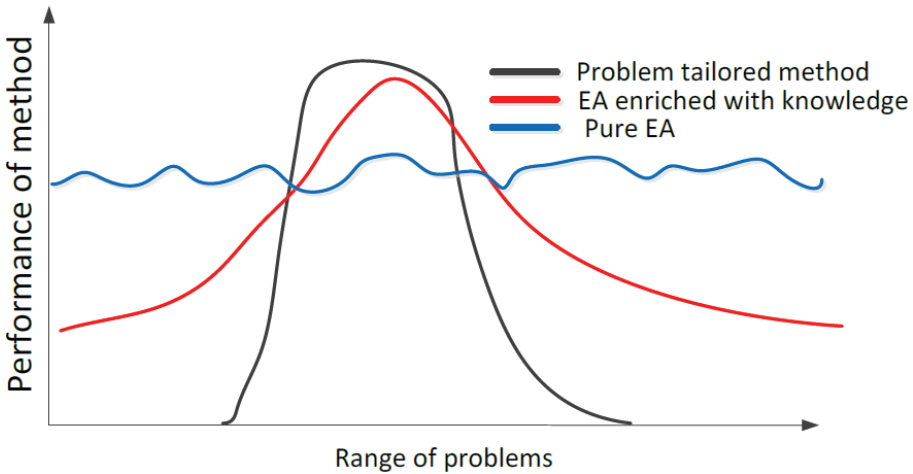


Fig. 31. The quality of results for different search algorithms in function of possible range of problems solved – cf. Michalewicz [129].

Memetic algorithms is a new tool for GPP. In fact there are only several publications with first attempts to use this class of computation for solving GPP. The basic structure of this framework is given below in Fig. 32.

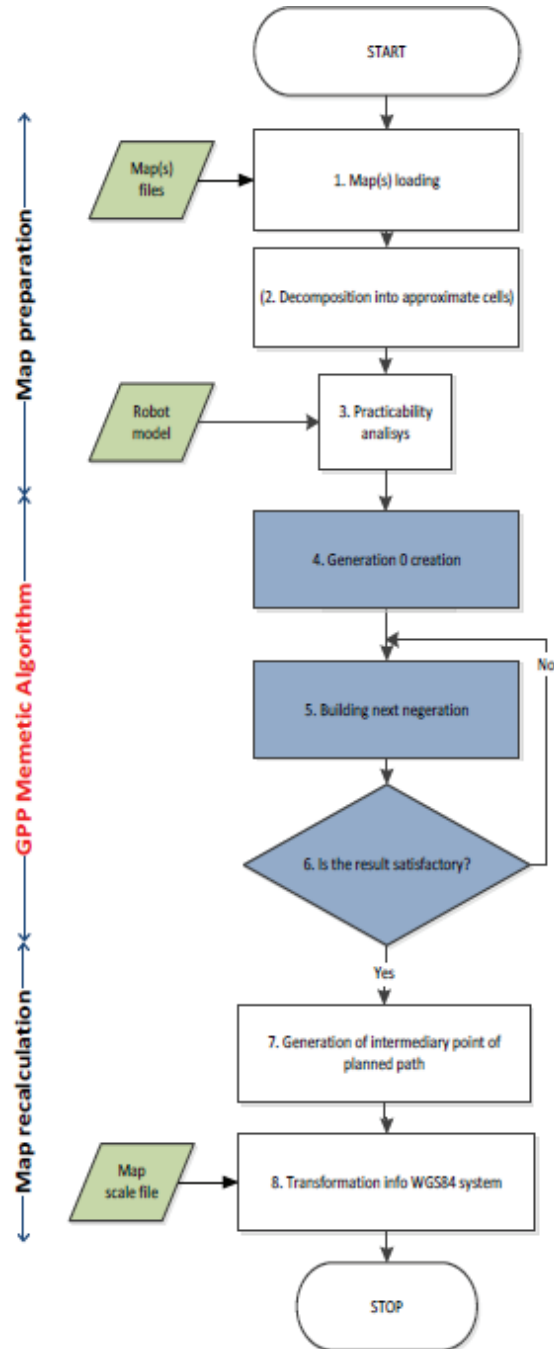


Fig. 32. The basic structure of the memetic algorithm for the global path planning for a mobile robot.

A basic candidate solution in the algorithm to be presented is a specimen of following form:

$$s = \langle A_s, P_s, f_s, N_s, L_s, V_s, O_s, U_s, \Delta L_s \rangle \quad (33)$$

- $A_s(s)$ – variable length genotype of a specimen,
- $P_s(s)$ – variable length phenotype of a specimen,
- $f_s(s)$ – value of the fitness function of a specimen, refer to f' of (14),
- $N_s(s)$ – number of segments of a specimen and, simultaneously, the number of alleles in a specimen genotype (path segments)
- $L_s(s)$ – length of a specimen in the map units
- $V_s(s)$ – vector of a path planning rule violation
- $O_s(s)$ – age of specimen counted in the number of generations
- $U_s(s)$ – vector of reparability of a specimen
- $\Delta L_s(s)$ – improvement sampling frequency (see Section 4.1.3)

The length $L_s(s)$ of a specimen is calculated as follows:

$$L_s(s) = \sum_{i=0}^{N_s(s)-2} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (34)$$

and it is calculated in the units of map distance, i.e. 1 unit equals 1 cell distance.

The calculations in the algorithm are entirely done in the approximate cell decomposition space (Step 2 in Fig. 32). It is connected with the ease of coding of the integer pairs of genotypes and phenotypes. A genotype is coded in the form of a sequence (permutation) of two dimensional vector movements called here segments of movement. Each segment is therefore a $[x, y]$ vector with a non-constant length $\sqrt{x^2 + y^2}$. Also the number of alleles in the genotype, i.e. the number of segments for each specimen, varies for specimens along the population. These two variables of each specimen: a variable segment movement $[x, y]$ and a variable number of segments $N_s(s)$ in the genotype are optimized during the algorithm run because for a particular path there is a minimum number of segments that represent this path without losing the path smoothness.

A genotype of each specimen is then represented by:

$$A_s = (M_0, \dots, M_{N_s-1}) \quad (35)$$

$$M_j = (x_j, y_j, v_j) \quad (36)$$

The coordinate system for M_j is also an approximate cell decomposition coordinate system and therefore $[x_j, y_j]$ are integer values. The value of v_j determines whether the segment violates any planning rules and can have the values:

$$v_j = \begin{cases} -1 & \text{for the violation that has not been identified} \\ -2 & \text{for segment that do not violate any rule} \\ > 0 & \text{corresponding to a positive value of violation in vector } V_s \end{cases} \quad (37)$$

The phenotype P_s of a specimen is also calculated in the approximate cell decomposition domain and has the following form:

$$P_s(s) = (R_0, \dots, R_{N_s(s)}) \quad (38)$$

$$R_j = \begin{cases} B_{GPP}, & j = 0 \\ \left(\sum_{i=0}^j x_i, \sum_{i=0}^j y_i \right), & j > 0 \end{cases} \quad (39)$$

This means that the phenotype is of length $N_s(s) + 1$ containing the beginning B_{GPP} ⁹ and the destination point E_{GPP} and including all intermediate points along the planned path.

Vector V_s is a rule violation vector where the number of specified elements corresponds to v_j given in (36):

$$V_s(s) = (V_{j0}, V_{j1}, \dots, V_{ws}) \quad (40)$$

$$V_{js} = (I_{Vjs}, B_{Vjs}, E_{Vjs}, T_{Vjs}, U_{Vjs}) \quad (41)$$

$$\begin{aligned} I_{Vjs} & \quad - \text{identification number corresponding to } v_i \text{ of } M_i \text{ in (36),} \\ B_{Vjs}, E_{Vjs} & \quad - \text{beginning and end point of violation in the decomposition domain,} \\ T_{Vjs} & \quad - \text{type of violation,} \\ U_{Vjs} & = \begin{cases} 0 & \text{for a nonrepairable violation} \\ 1 & \text{for a violation that can be repaired by memetic operators} \end{cases} \end{aligned} \quad (42)$$

U_{Vjs} refers to the fact whether a specified violation can be repaired by the implemented repair methods or not. Some violations, e.g. exceeding the total length of the planned path, have a very limited short repair method. A detailed description of repair operators is given in Section 4.1.3.

The fitness function that guides the progress during the memetic algorithm run is one of the most important factor from the point of view of the convergence speed. A proper definition of the fitness function is a key issue because the parent selection and survival selection mechanisms are based on the fitness function value. A proper construction of the fitness function becomes important in a situation when one need to conclude about the population diversity based on the fitness function value distribution¹⁰ across the population.

In the proposed algorithm a weighted sum of particular factors is taken as the fitness of a particular specimen and the problem is treated as a minimization problem so that the algorithm seeks the minimum value of $f_s(s)$:

⁹ Refer to (3).

¹⁰ The fitness value distribution is not the only factor that determines the population diversity but it is a good factor characterizing the diversity.

$$f(s) = \overbrace{i_L c_L L_s(s)}^{\text{part 1}} + \overbrace{i_c c_c C_s(s)}^{\text{part 2}} + \overbrace{i_o c_o O_s(s)}^{\text{part 3}} + \overbrace{c_{vm} \sum_{i=0}^w U_{Vis} + c_{vL} \sum_{i=0}^w d(B_{Vis}, E_{Vis})}^{\text{part 4}} \quad (43)$$

- $L_s(s)$ – length of path – cf. (34),
- $C_s(s)$ – specimen cost value depending on the map,
- $O_s(s)$ – specimen age expressed in the number of generations,
- U_{Vis} – reparability of particular path planning violations, cf. (42),
- d – distance between specified points (here the beginning B_{Vis} and endpoint E_{Vis} of violation),
- $c_L, c_c, c_o, c_{vm}, c_{vL}$ – weighting/normalizing factors that are calculated in each generation; all weighting factors are calculated so that they normalize the particular factors in the weighted sum to 1,
- i_L, i_c, i_o – importance factors for the particular fitness elements.

To summarize, the fitness function (43) takes into account:

- *Part 1 – Length* of the path in the decomposition units. The decomposition units are easily recalculated to real distance in meters and are directly proportional to the distance. The smaller the region to which the map corresponds to, the more linear this proportion is. In fact, for the areas of 50 km × 50 km and less it is safe to assume a linear proportion while calculating the real distances which results in an error of linearity ~3.4 %.
The c_L weighting factor is equal to and is equal to $\frac{1}{\max(L_s(s), s \in G)}$ the inverse of the maximum length of a specimen in the population so that $i_L c_L L_s(s) < 1$.
- *Part 2 – Cost* of the path. Each cell on the map has its corresponding cost belonging to the interval $<0; 1>$ where 1 means that the cell is unreachable. The cost of the specimen is calculated as the cum of the costs of the cells visited along the path. Therefore, the user specifying the cost map can include some property of terrain as an additional amount of fuel that is consumed during the transition via the cell or an unreachable area due to the lack of wireless communication and others. It is up to the user what the interpretation of the cost would be.
- *Part 3 – Candidate solution age*. This is expressed in the number of generations and is present in (43) because the algorithm has to maintain a constant change and new candidate solutions should be preferred. It also means that if within some generations a high valuable schema is not transmitted into a newly generated candidate solution, then the specimen of such a genotype should not survive.
- *Part 4 – Violations*. There are two factors responsible for the specimen rule violations. These are the total number of committed violations $\sum_{i=0}^w U_{Vis}$ and the total length of violations $\sum_{i=0}^w d(B_{Vis}, E_{Vis})$. As mentioned in Section 2.6 while solving constraint optimization problems even the worst, in sense of a cost, length, etc., candidate solution that does not violate any rule must have

a better fitness than any solution violating whichever planning rule (refer to (15)). Therefore, this should be secured by c_{vn} and c_{vL} in comparison to c_L , c_c , c_o . Note that the maximum value of the first part of equation (43): $i_L c_L L(s) + i_c c_c C(s) + i_o c_o O(s)$, has a maximum value of 1 for the longest, the least cost-effective and oldest candidate solution¹¹ in a particular generation. The value of $c_{vn} = 1$ is then sufficient since U_{vis} are cardinal numbers. Also, while violating the rule it is more preferable to have a violation that is the shortest possible as, in general, repairing a shorter violation needs less computational resources, and a shorter violation is therefore preferable over longer ones.

The c_{vL} is chosen to be $\frac{1}{\max(L(s), s \in G)}$ because then the term

$c_{vL} \sum_{i=0}^w d(B_{vis}, E_{vis})$ expresses the proportional part of the specimen genotype that violates the planning rules. Note that this part is always smaller than 1 and even comparing a specimen with a high number of short (in length) violations with a specimen that violates the rule once but for a long distance, the first will have a worse (higher) fitness value. More detail of how the violations are treated within the algorithm is given below. Another important issue is that not all violations in the planned path increase the value of $f_s(s)$. Only these that are unable to be repaired using the memetic upgrade and repair operators (Section 4.1.3) are taken into account. That is why the term $\sum_{i=0}^w U_{vis}$ is part of $f_s(s)$ and not of the total number of violations w .

For the crossover operator a classic one-point crossover is used due to its simplicity. The reproduction scheme is two-on-better-one meaning that two parents produce many offspring of which one with the best fitness is chosen to create a candidate for the next generation. Because the specimen's genotype is of a variable length [cf. (35)], then a proper choice of the crossover points is a key issue.

The task of producing a new offspring is put rather in the parent selection mechanism than in the crossover operator itself. The crossover works in a simple manner: an algorithm finds possible crossover points (Fig. 33c) of two selected parents (Fig. 33a and Fig. 33b). Each common point corresponds to two possible offspring (in Fig. 33 only 3 of 6, non-violating offspring are shown). All new offspring are compared by their fitness $f_s(s)$ (43), and the best of them is chosen as a candidate to the next generation. Such more "intelligent" – memetic construction of the operator is more likely to produce a non-violating offspring from violating parents. This is a very important and advantageous feature of this operator, because a so-called recombination fitness step¹² becomes large and diminishes the performance of the operator.

¹¹ Such a coincidence of having the path that is simultaneously the longest, oldest and least cost effective have a near zero probability of occurrence. During the author's numerical experiments it has not occurred even once.

¹² The parameter describes which is the fitness value of an offspring compared with the fitness of parents. The value is positive when the offspring have a lower fitness value than the parents.

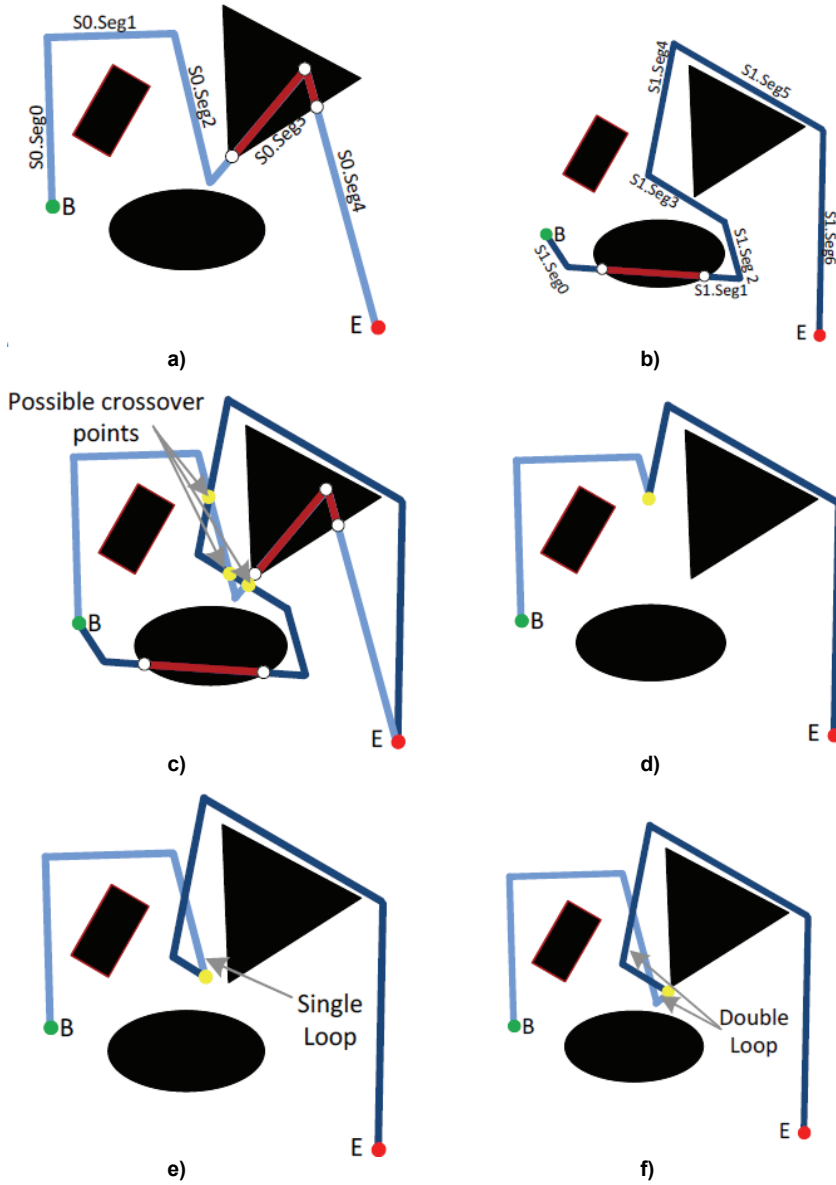


Fig. 33. An example of the one point crossover used in the proposed algorithm; a) specimen 0 – the first parent that is built of 5 segments of which two are a violation (Obstacle Encroach), b) specimen 1 – the second parent having one violation (Obstacle Encroach), c) two parents with the selected possible crossover points – common points of two specimen, d) the first offspring, e) the second possible offspring, f) the third possible offspring; legend: ● – path start point, ● – path destination point, ○ – path violation begin/end point, — – 1st specimen phenotype, — – violation along segments, — – 2nd specimen phenotype, ● – specimen cross points.

Parents having more than one common point will always produce an offspring containing loops. Nevertheless, these loops can be ousted by a loop removal operator (cf. Section 4.1.3). Such a crossover operator eliminates a situation when non-violating parents produce a violating offspring. And more: violating parents that recombine into a violating offspring produce new specimens that are not worse in their fitness than they themselves are. The two last sentences are crucial for understanding the power of this operator: it is guaranteed by the construction of the operator that the offspring produced by the parents will always be no worse than the parents' fitness – this guarantees a constant progress of the algorithm. Additionally to that intelligent crossover, the algorithm can also use a simple crossover with a randomly chosen point. The user can freely choose a proportion of how often to use both operators: on the one hand, the using of the intelligent operator only directs the genetic drift onto optimization path while, on the other hand, a random crossover point provides more search-like an approach. In fact, both schemes of recombination are used depending on the type of population as the proposed algorithm uses a multi-population optimization.

For the proposed algorithm the Simple Tournament Selection and Specialized Tournament Selection mechanisms are used in a proper proportion depending on the type of population.

The second basic genetic operator, *mutation*, is implemented in two forms:

- *Insertion mutation*. The chromosome is cut and a number of random segments are inserted between them. According to (39), in order to fulfill (13) while mutating any allele of the genotype, the last segment has to be modified (or added) to end the planned path in the destination point. The number of segments inserted ($N_{m,i,s}$) to the mutated genotype is subject to change and is a population parameter. It is usually assumed such that it maintains the golden rule of mutation successfulness at 0.2 [179]. Fig. 34 presents an example of an unsuccessful insertion mutation. The insertion mutation can be successful only if the segment following the mutation point (Fig. 34) is violating.
- *Deletion mutation*. It is an opposing operator to the insertion mutation. The number of segments that are placed after the mutation point are removed from the genotype. The number is also a population parameter and it should also maintain the golden rule of mutation successfulness at 0.2 [179]. The deletion mutation has a similar property to the insertion mutation concerning the change of the last segment.
- *Lengthen/Shorten segment*. This mutation consists in multiplying the segment length by a factor that has the normal distribution $N(1, 1/16)$. The standard deviation is kept constant during the algorithm run. This operator is symmetrical (the mean value of distribution equals 1) and hence there is no need to introduce any counter-balance operator.
- *Mirror Move mutation*. In this mutation the segment coordinates are multiplied by the vectors $[1, -1]^T$, $[-1, 1]^T$ or $[-1, -1]^T$ without changing the absolute values of the movement vector. The operator is also symmetric. Probabilities controlling the frequency of mirror move mutation are constant.

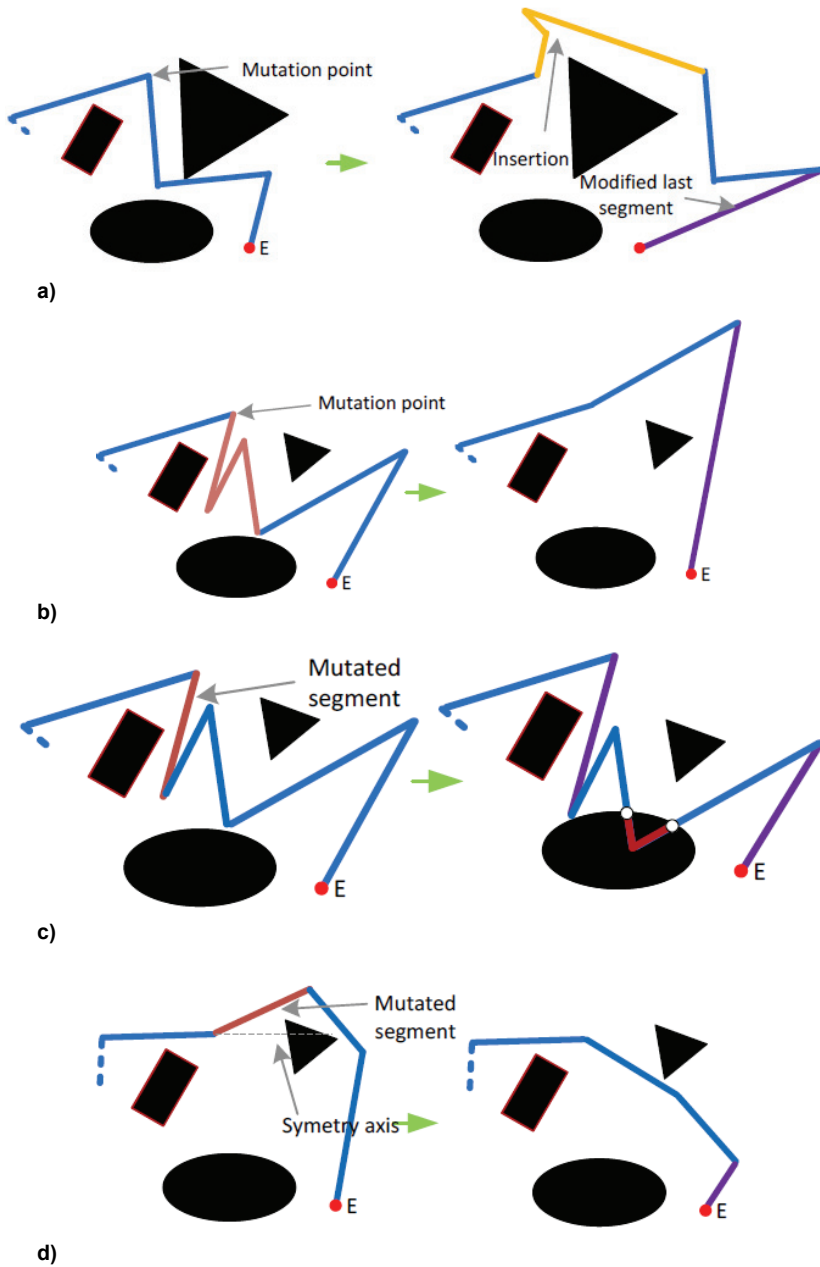


Fig. 34. An example of (a) the insertion mutation, (b) the deletion mutation, (c) the lengthen/shorten segment mutation, (d) mirror move mutation; legend: ● – path destination point, — specimen's phenotype, — new (added) segments, — remaining part of path (previous and next segment) begin/end point, — modified segments.

All search optimization and machine learning problems with constraints need a tool to deal with violations. In some cases violations can be easily overcome thanks to a specific specimen representation or a decoder functions [180–182] but in most cases this is impossible since decoder functions become complicated in higher order configuration spaces.

The last term of (43):

$$\overbrace{c_{vn} \sum_{i=0}^w U_{Vis}}^{\text{part 4.1}} + \overbrace{c_{vL} \sum_{i=0}^w d(B_{Vis}, E_{Vis})}^{\text{part 4.2}} \quad (44)$$

is known in the literature as the penalty function (punishment function) and is a well-documented and investigated way of dealing with the violation of constraints in the *COP* (constrained optimization problems).

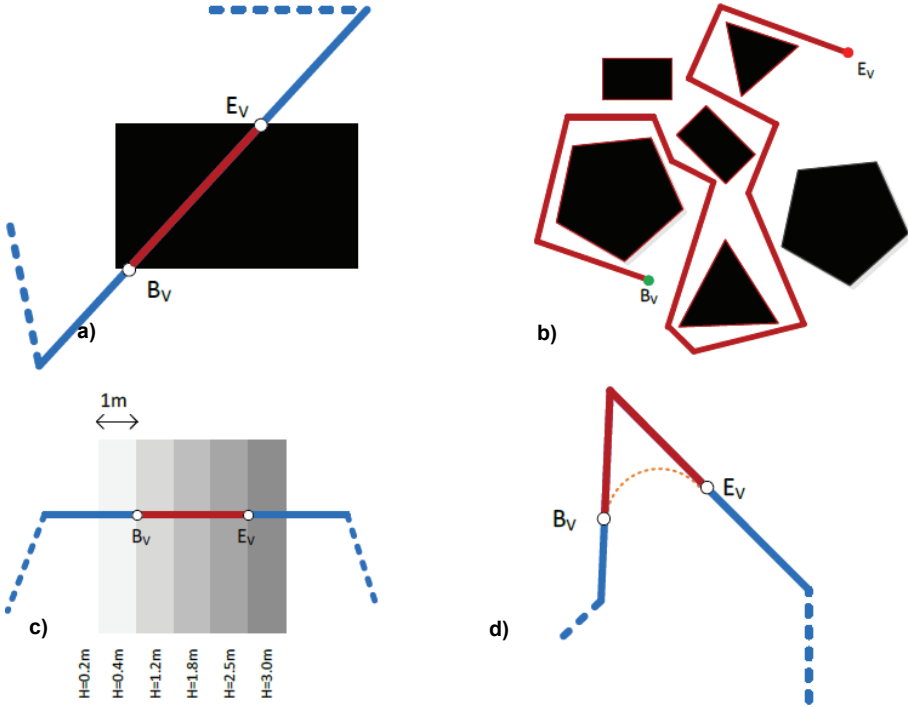


Fig. 35. Typical examples of planning violations for the proposed memetic algorithm for the GPP. In all schematic drawings binary parts of the maps are used for simplicity of presentation (see Legend below), a) Obstacle Encroachment, b) Too long a path, c) Abrupt uphill, d) Path smoothness violation; legend: $\square$ – free space (cost = 0.0), $\blacksquare$ – occupied space (cost = 1.0), — – path, --- – remaining part of path (previous and next segment), --- – violating part of path, --- – minimum turn circle, $\circ$ – begin (B_V) and end (E_V) points of violation, $\bullet$ – destination point, $\bullet$ – start point.

Evolutionary computation algorithms need proper tools to deal with violations. In the proposed algorithm the number of violations and the total violation length are compared. While the first (violation number: part 4.1 of (44)) carries the fitness value over 1, the latter (violation length: part 4.2 of (44)) serves for the comparison between different specimen of the same number of violations. It has been chosen that all violation types are treated equally without putting any emphasize on any violation types.

The following violation types are considered within the proposed algorithm:

- *Obstacle Encroach violation* (Fig. 35a). A very basic violation that occurs when a planned path leads the robot via an area that is occupied by an obstacle, i.e. that has the cost C equal to 1. Each transition between the non-1 to 1 cost is detected by the algorithm as the beginning of the obstacle encroach violation. The length of violation is calculated as the Euclidean distance between B_V and E_V – the beginning and endpoint of violation occurrence. If the segment ends inside the region occupied by the obstacle, E_V is set to be the last point of the segment and this results in counting the violation twice: for the current and following segment.
- *Path Length violation*. This violation (Fig. 35b) is related to the Current energy level, Energy consumption from the properties of the robot model. If a path is longer than the robot energy availability allows, then the whole distance along the path violates the rule. This violation can occur only once, but the length equals the total length of the path. This error does not occur for the ideal point holonomic robot (due to the infinite fuel level).
- *Hill abrupt violation*. If the steepness of the hill that the path approaches exceeds the maximum hill angle coverage parameter, a part of the segment where this violation occurs is considered to be violating. Note that a height map is needed for calculating this type of violation and this calculation may be complex. This error does not occur for the ideal point holonomic robot.
- *Path smoothness violation*. If the particular segment connection produces a turn the angle of which exceeds the minimum turn curvature parameter, such segments part are considered as violating. In the case that the turn angle is too sharp, there are two possibilities (Fig. 36, Fig. 37). The robot turns before the segments meeting point with the minimum turn angle so that the circle is inside a concave angle while the planned segments are tangent to this circle. If the movement circle passes through the obstacle points of osculation of circle and planned segment become the beginning B_V and the end point E_V of the violation (Fig. 36). In the latter case the movement is realized outside (Fig. 37) of the concave angle. The robot starts to turn in the common point of two segments and moves along the circle curvature of the minimum radius until a return is possible. A return point (Fig. 37) is a point that belongs to the second circle of movement and is tangent to the first and the second segment of movement. If such a two-circle movement is impossible due to the *Obstacle Encroach Violation*, then the *Path smoothness violation* occurs. For the interior circle the centre point is described by:

$$x_m = \frac{r_{\min} (\sqrt{a_1^2 + 1} - \sqrt{a_2^2 + 1}) \pm b_2 \pm b_1}{\pm (a_1 \pm a_2)} \quad (45)$$

$$y_m = r_{\min} \sqrt{a_1^2 + 1} + b_1 + a_2 \frac{r_{\min} (\sqrt{a_1^2 + 1} - \sqrt{a_2^2 + 1}) \pm b_2 \pm b_1}{\pm (a_1 \pm a_2)} \quad (46)$$

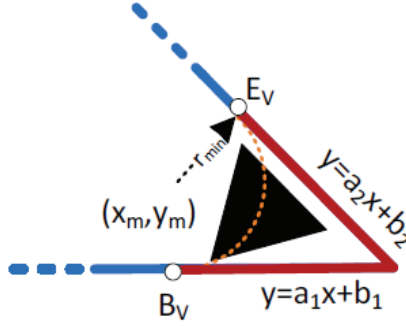


Fig. 36. The first possibility for taking the turn in the interior part of a concave angle by a circular movement of $r_{\min}$ radius. Depending on the angle value, the circle centre point is closer or farther from the binding point of two segments; legend: — path, - - - – remaining part of the path (the previous and the next segment), — – violating part of path, - - - – the minimum turn circle, $\circ$ – beginning (B_V) and end (E_V) points of violation.

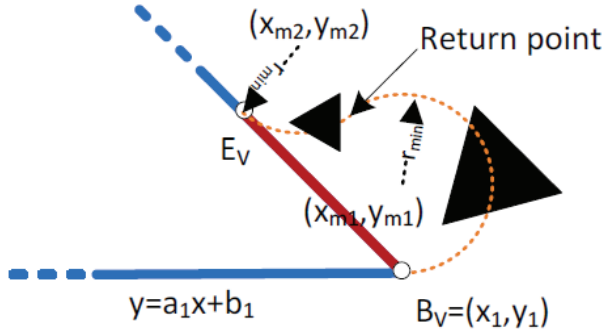


Fig. 37. The second possibility for taking the turn in the exterior part of a concave angle by two circular movements of $r_{\min}$ radius; legend: — path, - - - – remaining part of the path (the previous and the next segment), — – violating part of path, - - - – the minimum turn circle, $\circ$ – beginning (B_V) and end (E_V) points of violation.

For taking the turn outside of the concave angle, the robot turns using two circle movement the centre points of which are described by

$$x_{m1} = \frac{r_{\min} \sqrt{a_1^2 + 1} \pm y_1 \pm \frac{1}{a_1} x_1 \pm b_1}{\pm \left(a_1 \pm \frac{1}{a_1} \right)} \quad (47)$$

$$y_{m1} = -\frac{1}{a_1} \frac{r_{\min} \sqrt{a_1^2 + 1} \pm y_1 \pm \frac{1}{a_1} x_1 \pm b_1}{\pm \left(a_1 \pm \frac{1}{a_1} \right)} + y_1 + \frac{1}{a_1} x_1 \quad (48)$$

$$x_{m2} = \pm \frac{r_{\min} \sqrt{a_2^2 + 1} \pm r_{\min} \pm x_{m1} \pm y_{m1} \pm b_2}{a_2 \pm 1} \pm r_{\min} \frac{a_2}{\sqrt{1 + a_2^2}} \quad (49)$$

$$y_{m2} = \pm \left(\sqrt{r_{\min}^2 - \frac{r_{\min} \sqrt{a_2^2 + 1} \pm r_{\min} \pm x_{m1} \pm y_{m1} \pm b_2}{a_2 \pm 1} \pm y_{m1}} \right) \pm r_{\min} \frac{1}{\sqrt{1 + a_2^2}} \quad (50)$$

The equation for the centre points of the circle come from simple geometric calculations on a plane whilst the $\pm$ signs come from the fact that the values have to be absolute.

To summarize, there are four main violation types recognized by the algorithm: *Obstacle Encroachment*, *Path Length*, *Hill Abrupt* and *Path Smoothness* violations. In fact, the framework of the algorithm proposed in this thesis makes it possible to easily extend the definitions for other types of violations by adding violation definitions that include: an occurrence condition, B_V , E_V calculation (for the last term in (43)) and violation repair methods. The remaining part of an algorithm does not change.

4.1.3. Improvement and repair methods for GPP

A genotype of a specimen can include movements that are either violating or redundant and that have to be ousted or repaired. The repair and ousting are implemented as mutations and lead to an increase in the fitness by reducing the length of a specimen, and by the elimination of violations, and are leading memetic properties of an algorithm. The *improvement and repair* methods work in a different way than the mutation and crossover which fulfils the postulate of Krasnogor [177] to introduce new operator types that increase population diversity. All the improvement type methods aim at decreasing the length of the specimen but they also take into account the cost of an improvement. In some case insufficient changes are accepted i.e. operators always improvement length produced but in some cases the cost influence of the path produced is enlarged. This is especially done to decrease the number of successful improvement steps by the operators for not losing the diversity. The author has noticed during the research that having only true

improvements via the improvement operator¹³, the algorithm is pushed to a local minimum and this leads to a drastic increase of mutation probabilities and lowers the speed of convergence¹⁴. The proposed operators are characterized by their successfulness ratio. It is counted by dividing the number of successful operator executions, i.e. the number of times when the operator terminates and the new solution found is better and the new genotype has a better fitness, by the total number of runs of the operator.

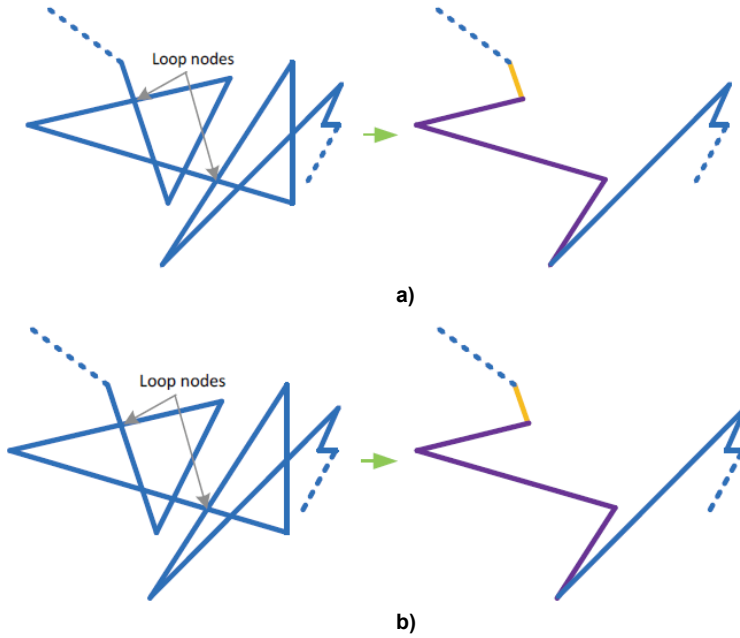


Fig. 38. Examples of possible methods of loop removal in the proposed algorithm; a) Exact Loop Removing Operator, b) Rough Loop Removing Operator; legend: ■■■ – path, ■■■■ – remaining part of the path (the previous and the next segment), ■■■ – new (added) segment(s), ■■■ – modified segments.

Redundant movements include loops and it turns out that can be simplified to straight movements. For removing a loop two methods are proposed:

- *Exact Loops Removal* (Fig. 38a),
- *Coarse Loop Removal* (Fig. 38b).

Loops are searched by calculating each straight segment equation on a plane and calculating nodes – the cross point for each segment with the remaining of segments. The *Exact Loop Removal* method detects a node of a loop, simplifies the

¹³ i.e. improvement in total specimen's fitness value, and not only a part of (43).

¹⁴ Note that even when the algorithm end up in the local minimum, the proposed memetic algorithm is self adapting and it will eventually find a way out of such a minimum, for instance by increasing the mutation rate.

loop to a single point and connects the previous segment (without the loop) with the node. After having removed the first loop, another loop is searched and the process repeats until all loops are removed. Such a movement is always possible when a genotype does not violate the *Obstacle Encroach* rule because the planned path proceeds along the same trajectory. The success rate of this operator¹⁵ is equal to 100 %. On the other hand, the *Coarse Loop Removal* finds the beginning and endpoint of segments of the loop or consecutive loops and connects them if such a movement does not violate any rule. The success rate for this operator is equal to 100 % only if no obstacle is present on the map, and the higher the density of obstacles, the lower the success rate. Nevertheless, path parts produced by the Rough Loop Removal operator are shorter and generally more cost effective when they do not violate any GPP rule.

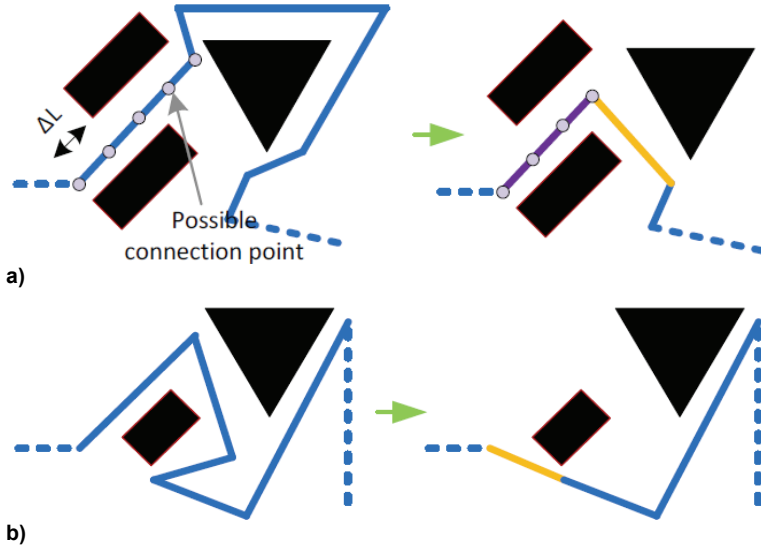


Fig. 39. Examples of possible methods of segments shortening in the proposed algorithm; a) Exact segment shortening with a specified factor ΔL_s refer to (33), b) Rough segment shortening; legend: — path, — — — — remaining part of the path (the previous and the next segment), — — — — new (added) segment(s), — — — — modified segments, ● — sampling points.

Redundant moves can also be simplified by shortening which is similar to the loop removal, and can be in two forms:

- *Exact Segment Shortening* (Fig. 39a),
- *Coarse Segment Shortening* (Fig. 39b).

¹⁵ It is measured by the ratio of the number of loops that do not violate the planning rules which can be simplified with the operator by the total number of loops that do not violate the planning rule.

The *Exact Segment Shortening* searches for all segment endpoints. For each i -th endpoint it tries to find $(i+2)$ -th endpoint that it can connect to without braking any planning rule. The time complexity of this operator is the same as that of the *Coarse Loop Removal* but the success rate does not equal 100 % and depends on the map content. The *Exact Loop Removal* samples each segment with a specified ΔL_s [Fig. 39d and (33)] frequency and checks whether there is a possible, non-violating connection with an endpoint of another further segment. ΔL_s is chosen by the successfulness ratio: starting with the value ΔL_s corresponding to 1/5 of the segment length for a new specimen. If the successfulness ratio exceeds the threshold of 25 %, it is increased to 1/4 of the segment length and if its value remains over 25 %, then it is increased more. On the contrary ΔL_s is decreased to 1/6 if success rate remains below 15 % and is decreased to 1/6, 1/7, 1/8 and this is continued until the state does not change. ΔL_s is characteristic to a particular specimen and is not inherited.

The selection of a proper ΔL_s is a key issue for this operator because having a small ΔL_s leads to finding better shortenings but simultaneously a smaller ΔL_s implies that the algorithm has to perform more samplings, and even more – too small ΔL_s forces algorithm to choose a shortcut earlier which is less optimal in general because the *Exact Shortening Operator* is greedy.

The *Coarse Segment shortening* seeks for a possible shortcut that does not violate any planning rule between the endpoints of segments and connects them. Note that in the case of successful ratio of the *Exact Segment Shortening* operator maintaining its level over 25 % for more than 5 generations, the functioning of the *Exact Segment Shortening* and *Rough Segment Shortening* operator is identical.

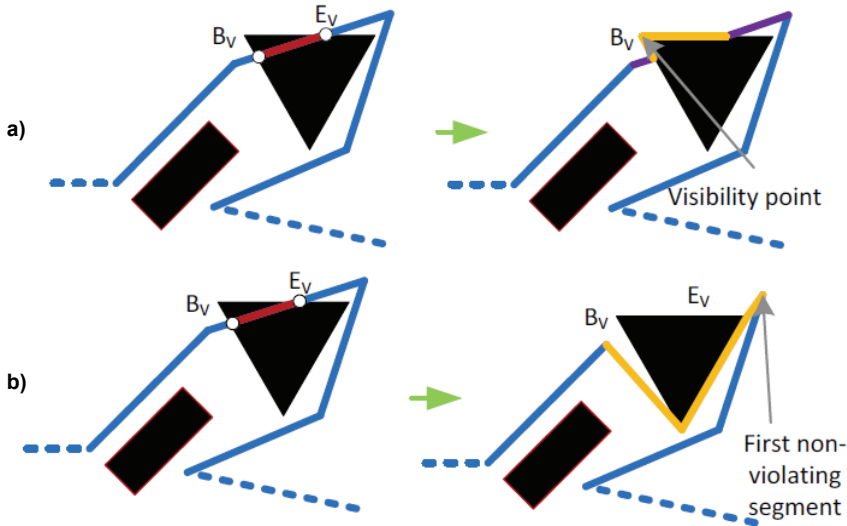


Fig. 40. Examples of possible methods of violation repair; a) *Exact Graph Repair*, b) *Coarse Graph Repair*; legend: — – path, - - - – remaining part of the path (the previous and the next segment), — – new (added) segment(s), — – modified segments, — – violating part of path, $\circ$ – beginning (B_v) and end (E_v) points of violation.

In addition to the improvement operators, other methods of a memetic type have been implemented in the proposed algorithm. If a segment or a group of segments encounter a violation of the *Obstacle Encroach* or *Abrupt Hill*, they can be repaired using a local graph. During the Practicability Analysis (see Step 3, Fig. 32) a graph is built. It is done by simply finding shapes of the vertices by comparing morphological elements of the 3×3 , 4×4 and 5×5 size with the map and selecting those that match vertex patterns. Next, a visibility between these elements is found taking into account the *Obstacle Encroach* and *Hill Abrupt* violation. Such a graph, similar to the Visibility Graph (Section 3.1.1) is then used for repairing the specimen's genotype and is called a *Repair Graph*. The edges have weights that equal the distance plus a cost to mirror their future value in the fitness function (43). The violation repair is then made using two methods:

- *Exact Graph Repair* (Fig. 40a),
- *Coarse Graph Repair* (Fig. 40b).

In *Exact Graph Repair*, vertices of Repair Graph are supplemented by two points: the point “just before” B_V and the point “just after” E_V ¹⁶. Then, the visibility of them is calculated with remaining vertices of the *Repair Graph*, and the *Dijkstra Algorithm* is used to find a traverse through the complete *Repair Graph*. If such a traverse exists, then a genotype of the specimen is updated.

In the case of *Coarse Graph Repair*, points that complement the Repair Graph are the beginning point of the first segment containing the violation and the endpoint of last segment containing the violation. The probabilities of graph repair have the same values as for the mutation operators. In this work we do not analyze the efficiency of employing particular graph search technique, but rather we consider a benefit of combining memetic algorithm with local graph that is implemented as genotype repair operator.

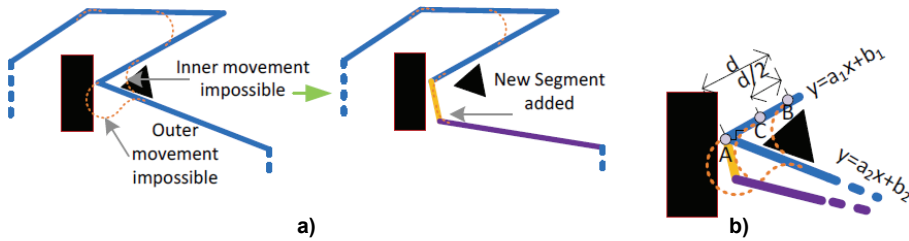


Fig. 41. Example of possible methods of violation path smoothness repair; legend: — path, ■■■■ — remaining part of the path (the previous and the next segment), — new (added) segment(s), — — — — the minimum turn circle, — — — — modified segments, ○ — sampling points.

The *Path Smoothness Violation* is repaired by a type of insertion mutation. An $\overline{AB}$ segment between the first and last circle (Fig. 41b) of length $|\overline{AB}| = d$ (Fig. 41b) is divided into two on $y = a_1x + b_1$ and in point C a new tangent to the $y = a_1x + b_1$

¹⁶ Just before and just after refer to the direction of movement.

segment is drawn. If the new pair of segments, both tangent to the circle originating from , do not violate the smoothness rule, then they replace the old $y = a_2x + b_2$, otherwise $\overline{AC}$ and $\overline{CB}$ segments are divided into two of $d/4$ length producing points D and E . The new circles are drawn for D and E points and for each circle a pair of segments is checked if they violate the Obstacle Encroach rule. The process is repeated until a pair of segments is found. In practice the author limited the depth of the performed bisection to 5 producing the smallest possible segment of length $d/32$. If within 5 trials no valid segment pair can be found, then a violation is considered to be impossible to repair (refer to (42)) and such a violation increases the value of $f_s(s)$.

The improvement operators are implemented in such a way that they work with some probabilities. $p_{o,r}$ is the probability that a particular specimen will be repaired while $p_{o,r,s}$ is the probability that a segment/loop within a chosen specimen will be improved. $p_{o,r}$ should be kept low so that approximately only 5 % of the population is considered for improvement, while $p_{o,r,s}$ should be as high as 0.6¹⁷. No influence of both improvements on the population diversity has been noticed – there is also no empirical rationale for such a dependence. This is explained by the fact that the result of the operators, a improved genotype, is dependent on its previous, not improved form so that the improvement can be considered as a translational vector without specified coordinates in the S-space. These probabilities can be kept constant without back dependency of the population diversity on them.

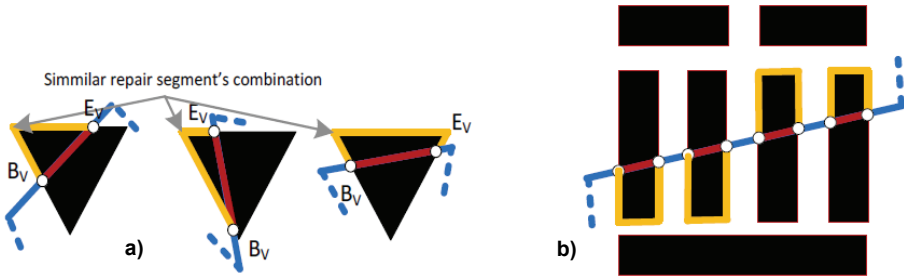


Fig. 42. a) Example of different Obstacle Encroach violations repaired with a similar segment combination, b) The risk of repairing the segment with the Exact Graph Repair operator; legend: — path, — remaining part of the path (the previous and the next segment), — new (added) segment(s), — modified segments, ○ – beginning (B_v) and end (E_v) points of violation.

However, the *Graph Repair Operators* should be used in a more intelligent way. Fig. 42 shows a situation when different *Obstacle Encroach* violations are repaired in a very similar manner. If more specimen encroach obstacle, the same obstacle Graph Repair would produce a segment sequence within the genotype that is non-violating and therefore possibly attractive, being able to build up dominating schema in the population. The situation becomes dangerous if a map contains very

¹⁷ These are values for generating population G_0 .

many obstacles which are similar in shape; such maps are common in highly urbanized terrains (Fig. 42b). This situation does not occur for the rough graph repair operator.

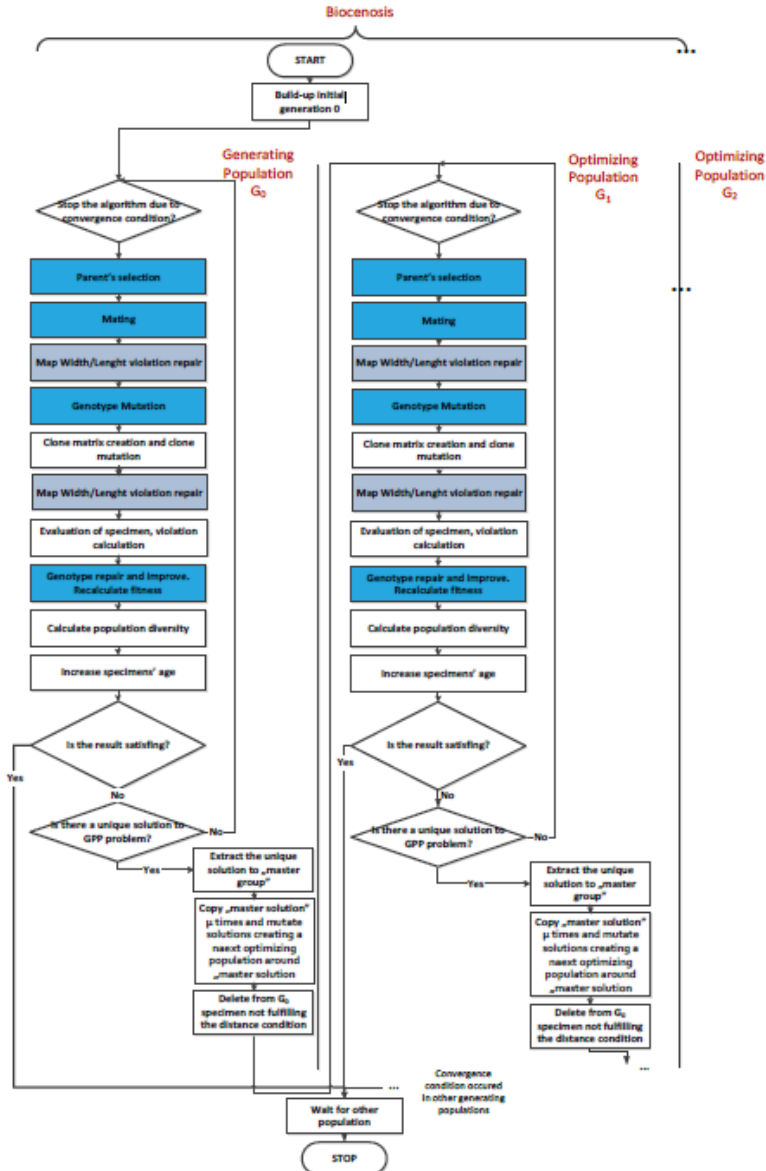


Fig. 43. The proposed memetic algorithm for the GPP. The blue color corresponds to memetic extensions of the genetic algorithm – color intensity corresponds with higher or lower share of memetic part. Only two simultaneous populations are shown: the one generating G_0 and the one first optimizing G_1 .

In the classic genetic algorithm the local search techniques are not employed for genotype repair. While using them there is an issue of how the repair result is included in the fitness function calculation. In practice there are two approaches: the Lamarckian Approach [183] and the Baldwin Approach [184], or a mix of the two. In general, the Lamarckianism claims that the genotype of a specimen after the local search is replaced by a new one – the one that was created by the local search. On the other hand, Baldwin says that the genotype should remain in its unaffected form but the specimen fitness should be updated. Hybrid methods update the fitness for all specimens after the local search but change the genotype only with some probability. As a result of many comparisons [151, 183, 185–187], the author's experiments and the current trend to use the pure Lamarckianism or a hybrid solution, the Lamarckian approach is used in the proposed algorithm.

In the figure shown below (Fig. 43) a detailed block diagram of the algorithm is depicted. The blue color indicates parts of algorithm in which the memetic approach was used. For the simplicity and ease of understanding the algorithm was drawn only partially: the development of two parallel populations is presented while the number of such populations is dynamically changing during the algorithm run. An unique property of the proposed algorithm is what will be called from now on a *Parcellment* of the population. This mechanism speeds up the convergence of the algorithm and brings a tool for specializing population properties for reaching different goals. This mechanism is biologically inspired by *Specie Specialization* that occurs in nature and increases and saves biodiversity.

The population *Parcellment* is a process of forming a completely new population during the algorithm run when a highly valuable specimen s_k is found within a current generation. Therefore, by finding a specimen s_k for which $\varphi_r(s_k) = 0$ (refer to (9)) within population G_n that is highly valuable and unique¹⁸, a new population G_k is produced. G_k is a population concentrated around s_k in terms of the S-space. It is formed by mutating specimen s_k until $\mu - 1$ unique specimens are found. These specimen together with s_k create population G_k that from this moment on becomes parallel to all already formed populations. An algorithm processes then several populations simultaneously, and a group of these populations is called *Biocenosis*. Simultaneously, specimen s_k , a master solution, belongs to the Master Group. The Master Group is a group of specimen around which new populations are formed:

$$M = (s_{k0}, s_{k1}, \dots, s_{k(n-2)}) \quad (51)$$

n – the current number of different populations in *Biocenosis*.

A solution s_k that produces a new population has to be:

- *Highly Valuable* specimen. Highly valuable equals to a condition of $\varphi_r(s_k) = 0$ meaning that this solution does not violate any planning rule. It also means that $f_s(s_k) < 1$ (part 4 of (43) equals 0). In terms of the Global Path Planning, to find such an s_k means that there is a possible path that omits obstacles in a specific way which can be planned for the assumed model of the robot.

¹⁸ High value and uniqueness is defined later in the text.

From another perspective, the perspective of a search task – the problem of finding s_k equals to the determination of a point in the S-space that is close to some local minimum and therefore we need to apply some optimization algorithm on s_k to find that minimum.

- *Unique Specimen* around all existing populations. Candidates for the *Master Group M* need to be unique and each candidate before it would imply that the population *Parcellment* should be checked for this uniqueness. A solution s_u is called unique if:

$$\forall s_k \in M (k \neq u) d_s(s_u, s_k) = \frac{2 \sum_{i=0}^{N_{TRIPLETS}-1} A_i}{L_{s_k} + L_{s_u}} > d_{th} \quad (52)$$

- d_{th} – is a threshold value that depends on the size of the map grain; for a proper cell decomposition (Section 1) it should be equal 50.0–60.0¹⁹,
- $d_s(N_{suj}, N_{skj})$ – the distance between two consecutive segment points of specimens under comparison,
- $N_{TRIPLETS}$ – the number of triplets (see Step 2 below).

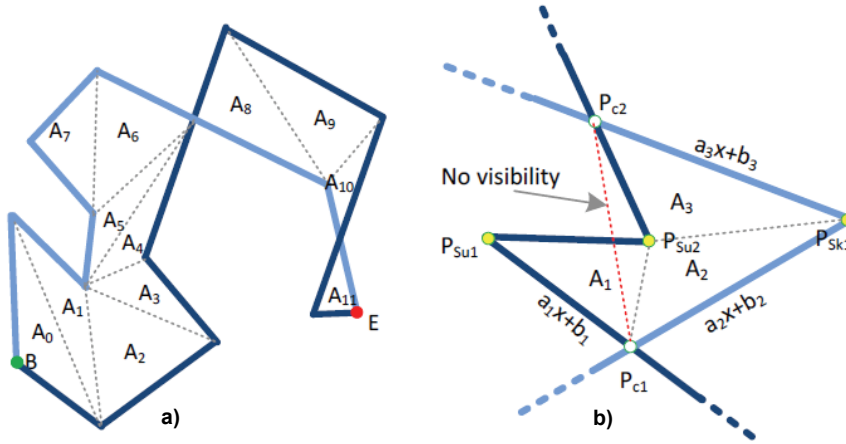


Fig. 44. a) A graphical representation of $\sum_{i=0}^{N_{TRIPLETS}-1} A_i = A_0 + A_1 + \dots + A_{11}$ – the distance between two specimen genotypes, b) A detailed drawing of $TRIPLET_{LIST}$ calculation.

The $d_s(s_u, s_k)$ is equal to the area between two broken lines, specimen genotypes. A graphical representation of d_s is shown in Fig. 44a. To speed up the calculation of the d_s distance we develop a fast method that works as follows:

¹⁹ On some maps the *Parcellment* is impossible as, e.g., it may be impossible due to the map size and obstacles exceeding d_{th} .

- Step 1.** Find all common points of the specimen's s_u and s_k genotype similar to finding points of a possible crossover (P_{c1}, P_{c2} in Fig. 44b). For each pair of points P_{c1}, P_{c2} find the endpoints $P_{Su1}, P_{Su2}, P_{Sk1}$ of the remaining segment and build two lists of points: $VERTEX_{LIST} = (P_{c1}, P_{Su1}, P_{Su2}, P_{Sk1}, P_{c2})$.
- Step 2.** Triplets of points are being found. For each point of $VERTEX_{LIST}$ check the visibility to every other point and if such a visibility exists, add it to $VISIBILITY_{LIST} = \{(P_{Su2}, P_{c1}), (P_{Su2}, P_{Sk1})\}$. The visibility cannot be added to the list if it crosses any segment of specimen s_u or s_k or if it is already on $VISIBILITY_{LIST}$. $VISIBILITY_{LIST}$ is recalculated to form $TRIPLET_{LIST} = \{(P_{Su2}, P_{Sk1}, P_{c1}, y = a_2x + b_2), (P_{Su2}, P_{Sk1}, P_{c2}, y = a_3x + b_3)\}$. $TRIPLET_{LIST}$ count is always one more than a count of $VISIBILITY_{LIST}$ and if visibility list is empty (e.g. A_{11} in Fig. 44a). The points belonging to triplets form a triangle. The segments equations (Fig. 44b) are given a priori since they have to be calculated for crossover operator.
- Step 3.** The areas A_1, A_2, A_3 (Fig. 44b) of triangles formed by the triplet points are calculated using the equations like²⁰:

$$A_1 = \frac{|a_1 P_{Su2} \cdot x + b_1|}{2\sqrt{a_1^2 + b_1^2}} \sqrt{(P_{Su1} \cdot x - P_{c1} \cdot x)^2 + (P_{Su1} \cdot y - P_{c1} \cdot y)^2} \quad (53)$$

- Step 4.** All areas of triangles are added and the expression $\sum_{i=0}^{N_{TRIPLETS}-1} A_i$ is calculated.

Because there are two types of populations within the algorithm, the population's parameters in *Biocenosis* are optimized for two different tasks: (1) to find new solutions, and (2) to optimize the existing solutions. Population G_0 called a generating population solution is responsible for searching for a new solutions while the remaining populations $G_1, G_2, \dots$ aim at optimizing the *Master Solution* a basis of which they were created. The generation of the Master Solution can occur from any population in *Biocenosis* but due to a fact that the population formed around the Master Solution is concentrated around one point in a S-space, then the probability of creating a new, unique solution from a concentrated group is very low²¹, and therefore it can be assumed that the *Parcellment* occurs only from G_0 . If condition (52) is not met for a specimen belonging to and one of the Master Solutions from the Master Group M , specimen has to be deleted from that population. Because of a steady genetic drift between the current and next generation of G_0 , the probability of the Master's schema H_k to occur in the next generation is high. The probability of the next *Parcellment* and creation of a new population would also be high. Keeping that in mind, the elimination of specimens of G_0 close to the Master solutions is a must in order to avoid the population duplication.

²⁰ One equation is given only for reference.

²¹ Although such a mechanism of *Parcellment* of the optimizing population exists in the presented algorithm, it has never happen during the tests of the author's algorithm.

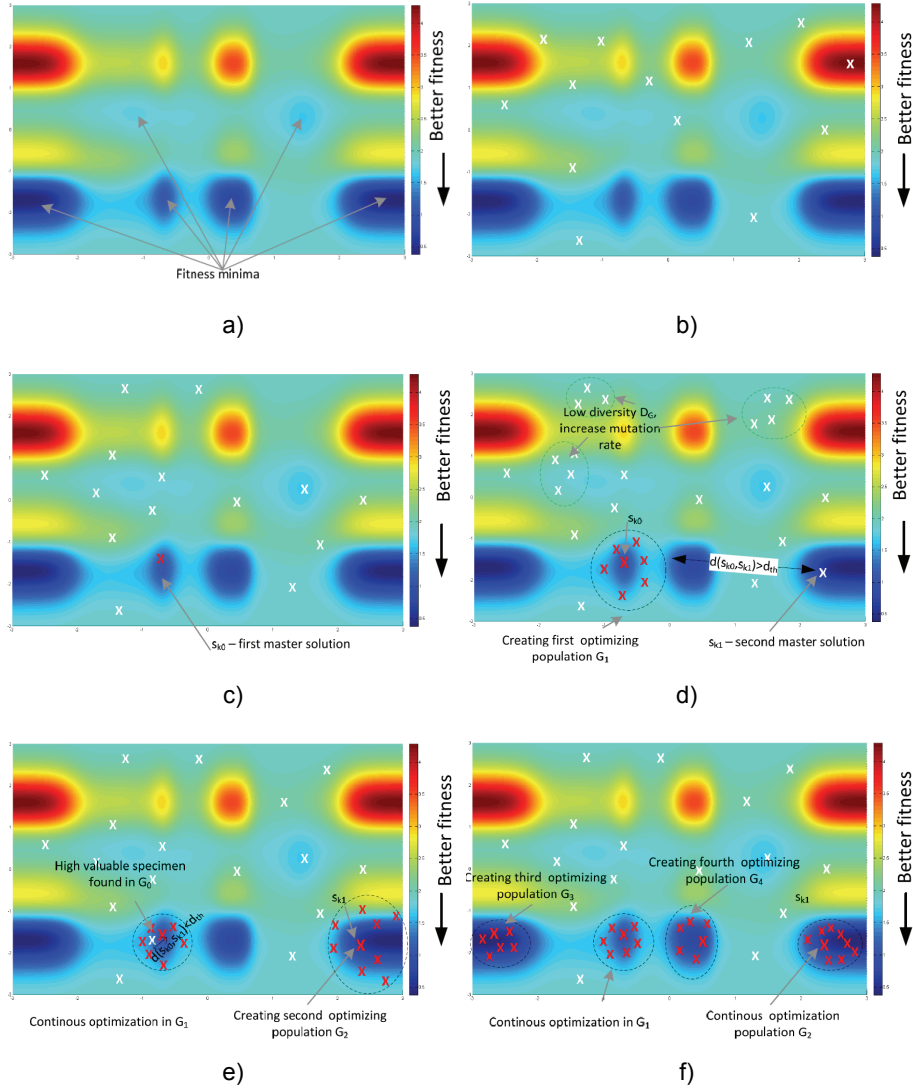


Fig. 45. Example of the algorithm run²².

In terms of search and optimization the algorithm of a mechanism of *Parcellment* within the *Biocenosis* can be depicted in the following way. The

²² The fitness landscape presented in this example is very simplified in order to show the order of phenomena occurring during algorithm run, therefore the search space is two dimensional (not true for the GPP problem), the fitness is a static function (not true for the GPP problem – refer to (43)), and the fitness coefficients are correspondingly equal to each other in different generations (not true for the GPP problem – refer to (43)).

primary population G_0 tries to cover the S-space in a way that the diversity of population is kept high. If a non-violating solution (Master Solution) is found, a new group (new optimizing population) of candidate solution around the Master Solution is built to cover the small region in the S-space in order to find a local minimum that may be a global minimum. A new optimized population is specialized in optimization so that it will eventually improve the Master Solution. Simultaneously, the G_0 search for another Master Solution. The theoretical algorithm run seen from the perspective of the S-space is depicted in Fig. 45 which also contains a description of sequence of steps of population development.

Using an intuitively appealing example, a generating population is a group of nomads going across the land (the S-space) and looking for water resources (a local optimum). If a water reservoir is found by one nomad (*Master Solution*), this nomad stays by the reservoir and because natural conditions are gentle in this area, he settles down, builds a village, reproduces and specializes in using resources of the reservoir (creates a new population and optimizes it). Simultaneously, all remaining nomads search for another reservoir but they keep the reservoirs found out of reach of the remaining nomads (G_0 searches for another *Master Solution*).

Figure 45 depicts a spread of populations in the proposed algorithm over the fitness landscape which is here some abstract two dimensional curve. Normally, its shape is not known, only the sampled values. a) The fitness landscape has six local minima drawn as blue/dark blue regions of the fitness landscape – an algorithm will try to find these minima, locally best solutions. b) A random population G_0 is created – specimens are indicated as points (white X) on the fitness landscape. c) After some generations a master solution s_{k0} (red X) is found for which the fitness value remains below 1. d) A population G_1 around s_{k0} is formed. Because the specimens belonging to G_1 are mutated from s_{k0} , the population is limited to some region of two dimensional search space. Simultaneously, another s_{k1} master solution is found and because it lies far away from s_{k0} ((52) is met), then it becomes a new master solution around which G_2 population will be formed. e) If a new highly valuable specimen of G_0 is found that is not unique, a new population is not created due to (52). f) After some generations all local fitness minima are found and the Biocenosis consists of five populations. G_1, G_2, G_3, G_4 are optimized until the convergence conditions are met (see Table 3).

The generating and optimizing populations are specialized in terms of having population parameters adjusted to the function they are meant to fulfill. These parameters are gathered in the table below (Table 3).

The power of the proposed approach is that the parameter values presented in Table 3 do not have to be exactly tuned to the map and because the algorithm is self-adapting. The table shows an example set only, though this can be modified. The modification leads to a faster or slower convergence but any value around those presented in Table 3 will work for any path and any map.

The example of specialization in the optimizing population over G_0 are seen in the memetic properties of the presented algorithm:

- Higher probabilities of the graph repair methods ($p_{o,r,CGR}$ and $p_{o,r,EGR}$).
- Higher probabilities of the improvement operators ($p_{o,i,ELR}$, $p_{o,i,CLR}$, $p_{o,i,ESS}$, $p_{o,i,CSS}$).

Table 3. A sample algorithm parameter values for the generating and optimizing population.

Parameter	Generating population G_0	Optimizing Population
Population size	$\mu = 50$	$\mu = 35$
Initial population	Random	Created as a mutation of the master solution
Crossover methods	One point crossover with no point selection – blind ($p_{0,c,b} = 0.7$) One point crossover with predefined points – intelligent ($p_{0,c,i} = 0.1$) Uniform crossover ($p_{0,c,u} = 0.2$)	One point crossover with no point selection – blind ($p_{0,c,b} = 0.15$) One point crossover with predefined points – intelligent ($p_{0,c,i} = 0.8$) Uniform crossover ($p_{0,c,u} = 0.05$)
Crossover Schema	Two-on-best-one	Two-on-best-one
Selection Schema	Simple Tournament Selection For the uniform crossover and blind one-point crossover, Specialized Tournament Selection for an intelligent crossover	Simple tournament selection For the uniform crossover and blind one-point crossover, Specialized Tournament Selection for an intelligent crossover
Initial number of Inserted Segments ($N_{m,i,s}$)	5	2
Mutation probabilities	$p_{0,m,s}$ for $c_{0,m,s} = 0.1$, $D_{Gth,s} = 1$ $p_{0,m,a}$ for $c_{0,m,a} = 0.3$, $D_{Gth,a} = 1$	$p_{0,m,s}$ for $c_{0,m,s} = 0.0$, $D_{Gth,s} = 0.5$ $p_{0,m,a}$ for $c_{0,m,a} = 0.0$, $D_{Gth,a} = 0.6$
Mutation schema	Insertion mutation/Deletion mutation ($p_{0,m,id}$), Lengthen/Shorten segment ($p_{0,m,ls}$), Mirror Move mutation ($p_{0,m,mm}$) Applied with equal probability of ($p_{0,m,id} = p_{0,m,ls} = p_{0,m,mm} = 1/3$ within $p_{0,m,s}$ and $p_{0,m,a}$)	Insertion mutation/Deletion mutation $p_{0,m,id} = 0.5$, Lengthen/Shorten segment $p_{0,m,ls} = 0.4$, Mirror Move mutation ($p_{0,m,mm} = 0.1$), All probabilities within $p_{0,m,s}$ and $p_{0,m,a}$
Population model	Only the best survive	Only the best survive with elitism
Starting ΔL	1/5	1/10
Improvement methods	Coarse Loop Removal ($p_{0,i,CLR} = 0.02$), Coarse Segment Shortening $p_{0,i,CSS} = 0.02$	Exact Loops Removal ($p_{0,i,ELR} = 0.05$) Coarse Loops Removal ($p_{0,i,CLR} = 0.05$) Exact Segment Shortening ($p_{0,i,ESS} = 0.05$) Coarse Segment Shortening ($p_{0,i,CSS} = 0.05$)
Repair methods	Coarse Graph Repair ($p_{0,r,CGR} = 0.01$)	Coarse Graph Repair ($p_{0,r,CGR} = 0.03$) Exact Graph Repair ($p_{0,r,EGR} = 0.03$)
Convergence condition	All optimizing population reached their convergence condition	Best specimen fitness over the last 15 generation not decreased more than 3 %

- A bigger variety of the improvement operators.
- A more accurate sampling in the improvement methods (ΔL).
- Elitism employed in the survival selection that guarantees to maintain optimization within the minimum found by the master solution.
- Lower blind mutation probabilities; this mechanism keeps the population within a specified region of the S-space around the master solution.

- A higher emphasis on an intelligent one-point crossover and a specialized tournament.
- All populations can be implemented as separate threads, therefore we obtain several different regions of S-space (Fig. 45) being explored at one with the S-space coverage density depending on S-space region quality (in sense of fitness value). This brings out the possibility of hardware implementations of up to several thousand populations simultaneously.

4.1.4. Numerical examples

In this experiment the simplest version of the algorithm is used. We consider a holonomic point robot with no movement constraints. The HugeMazeMap is an artificial binary map: the white color corresponds to the traversable area, i.e. the corridor floor, the black color represents the non-traversable regions, i.e. the walls.

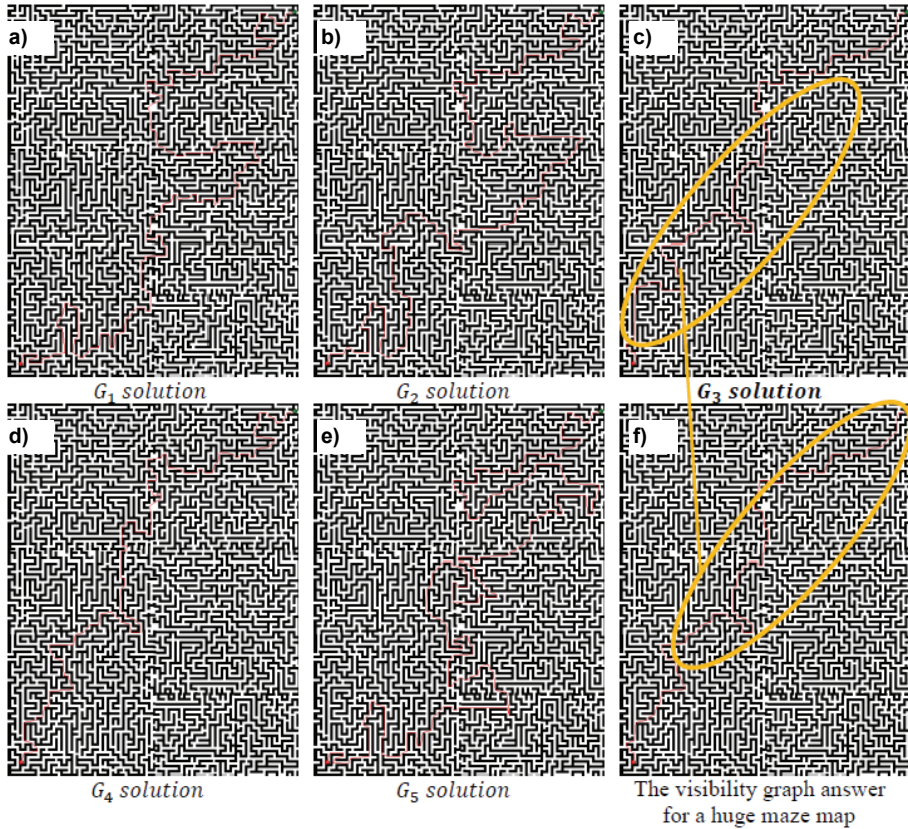


Fig. 46. Comparison of answer solutions for the GPP problem calculated by the proposed algorithm; a)–e) partial results obtained by the proposed algorithm; f) the GPP problem solved by the visibility graph algorithm for a comparison with the calculated solution [shown in c)].

Therefore, for the comparison of the specimen the violation vector (40), path length $L_s(s)$, and segment number $N_s(s)$ are used. There are at least 5 paths from the start to the destination point which lead to the creation of Biocenosis containing at least 5+1 populations.

During 106 generations 5 populations have been working in parallel while searching for the best path on the HugeMaze binary map. The summarization of progress in G_1 , G_2 , G_3 , G_4 , G_5 is quoted in Table 4 while the phenotypes are shown in Fig. 46.

Obviously generation G_3 produced best solution due to shortest length whilst the path cost C_s is equal zero for all other solutions. For binary where path length is the only one optimization criteria it is proofed (see Section 3.1.1) gives an optimal solution. A visibility graph based on Fibonacci heaps has been implemented during this research for comparison with proposed method. The result of this is presented in Fig. 46f. Visibility graph produced a path that is 1385.01, i.e. path calculated by proposed algorithm is only 4 % worse in length than optimal solution. Please note that a significant part of answer given by proposed algorithm is very similar to the one calculated by visibility graph²³ (Fig. 46, Table 4).

Table 4. A summarization of the planned global path. The path length, the number of segments, and the fitness function values are compared. The cost for all paths is equal zero due to the fact that the selected map is binary.

Population	Generation of genesis	Generation of convergence	Number of generations	$N_s(s_0)/$ progress ²⁴	$L_s(s_0)/$ progress	$f(s_0)$
G_1	2	50	48	8/72 %	1731.64/83 %	0.09
G_2	20	87	67	118/78 %	1831.14/77 %	0.03
G_3	34	88	54	86/72 %	1446.95/83 %	0.07
G_4	35	101	66	101/14 %	1684.62/20 %	0.05
G_5	51	106	57	106	3252.25/8 %	0.02
Visibility graph (Fibonacci heaps)	NA ²⁵	NA	NA	88/N	1385.01/NA	NA

In this case the calculated solution is not optimal as expected. Nevertheless, the algorithm shows a good performance, yielding a near optimal result, as proven by a comparison of path properties (Table 3) with the optimal visibility graph.

By analyzing the fitness function values (Fig. 47) over time expressed in generations one can clearly see another interesting feature that distinguishes the proposed algorithm from a regular genetic algorithm.

The theoretical shape of the average fitness in a regular genetic algorithm is a function similar in shape (cf. the dashed line in Fig. 47) to $C_1/t + C_2$ there C_1 , C_2 are some constants and t is the generation number. In the proposed algorithm such a shape will never take place. This is because of three mechanisms involved:

²³ Fig. 46 and Table 4 present these solutions as nearly identical but analyzing the numerical values of intermediate points of the path, their results differ from each other.

²⁴ Progress is measured in percentage of the final finale value to the initial value. The smaller this number, the bigger progress has been made.

²⁵ These properties do not apply to the visibility graph.

- *Parcellment* of population.
- Removal of specimens whose distance to one of master solutions is smaller than d_{th} according to (52).
- Dynamic fitness function.

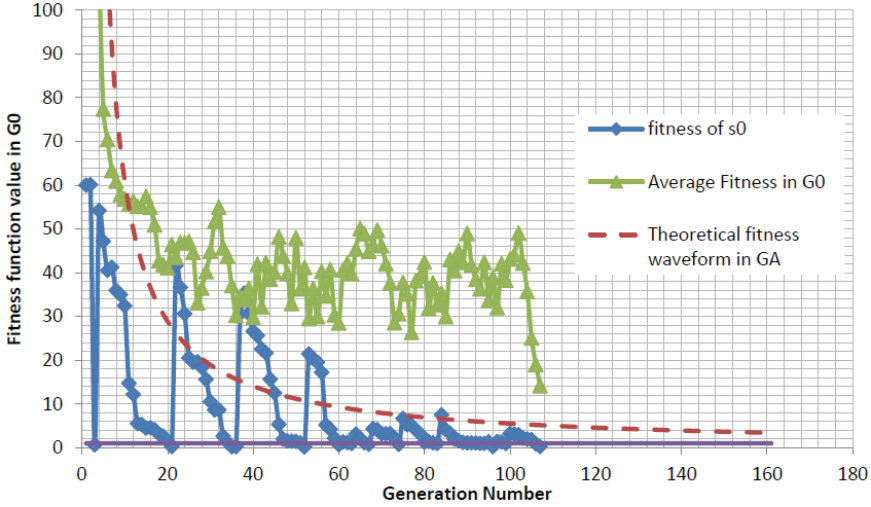


Fig. 47. Plot of the fitness function over generations in population G_0 .

The *Parcellment* of population leads to the removal a highly valuable specimen from G_0 . This implies a situation when the fitness value of s_0 increases to a number over 1. s_0 in generation $t+1$ is often specimen s_1 from generation t . Simultaneously, the average population fitness increases (refer to Fig. 47). Therefore, each time the *Parcellment* takes place, the average fitness of G_0 increases. In this example such a situation took happens in generations 2, 20, 35, 35 and 51 (cf. the average fitness function plot in Fig. 47).

The removal of specimens whose distance to one of the master solutions is smaller than d_{th} is similar in their influence on the fitness function value but a new population is not created this time and only a highly valuable specimen is removed.

A dynamic fitness function is the last reason for non-monotonic fitness functions. It is rather seen from the viewpoint of the average fitness value than the fitness of a single specimen. Each genetic algorithm based search guarantees not worse specimens in each next generation but only in the case when they are evaluated by the same fitness functions. Due to a specific construction of this function (43), the specimens have different fitness values depending on the current (average) situation in the population. Therefore. The fitness functions is not monotonic and can be different from population to population. The maximum difference between fitness function values of identical specimens belonging to different populations

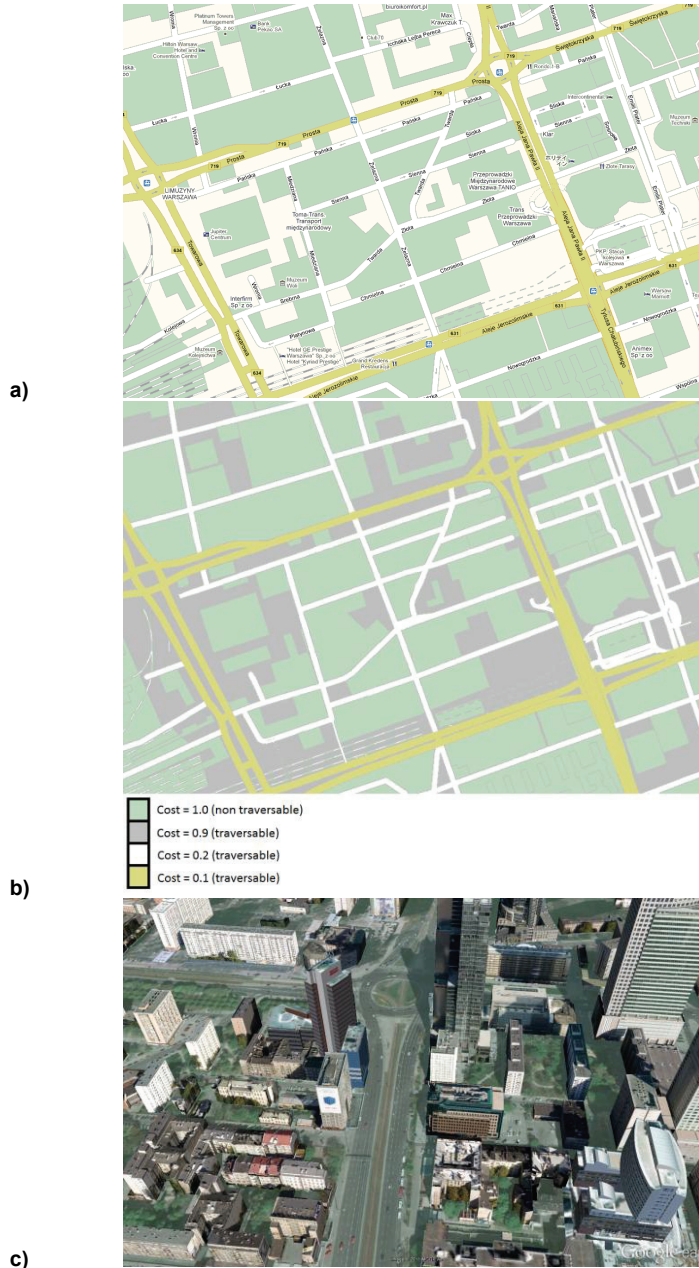


Fig. 48. The Warsaw map used in the computational experiment. Four cost levels are assumed and it is preferable for the robot to go along the streets (cost 0.1, 0.2) rather than through free space between buildings (cost 0.9); a) The source map downloaded from Google Maps; b) A raster map showing areas of an uniform cost. c) A 3D view of part of the map for the tilt angle calculation.

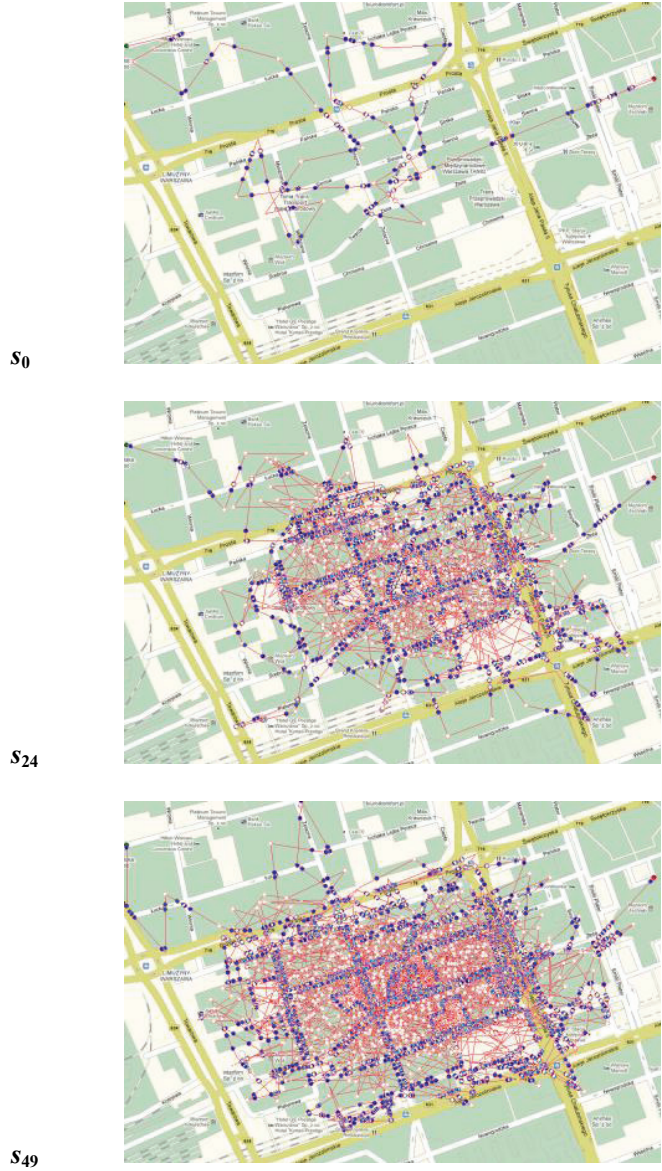
$$\left| \overline{f_{s,f+1}(t+1)} - \overline{f_{s,f}(t+1)} \right| < \mu \quad (54)$$

- $\overline{f_{s,X}(Y)}$ – average fitness in generation Y calculated with weighting/normalizing factors according to the X-th generation properties,
 μ – the number of specimens in the population.

In the case of the proposed algorithm, the average fitness can have a higher value of 50. And even more: even though there is a progress in population at $t + 1$ comparing to t , the average fitness value can increase due to changes in the weighting/normalizing coefficients of equation (43).

Table 5. Comparison of the genetic algorithm with the proposed memetic algorithm. The differences are shown in bold.

Parameter	Generating population G_0	
	Proposed memetic algorithm	GA
Population size	$\mu = 50$	$\mu = 50$
Initial population	Random	Random
Crossover methods	One point crossover with no point selection – blind ($p_{0,c,b} = 0.7$) One point crossover with predefined points – intelligent ($p_{0,c,i} = 0.1$) Uniform crossover ($p_{0,c,u} = 0.2$)	One point crossover with no point selection blind ($p_{0,c,b} = 1$)
Crossover Schema	Two-on-best-one	Two-on-best-one
Selection Schema	Simple Tournament Selection For the uniform crossover and blind one-point crossover, Specialized Tournament Selection for an intelligent crossover	Simple Tournament Selection
Initial number of Inserted Segments ($N_{m,i,s}$)	5	5
Mutation probabilities	$p_{0,m,s}$ for $c_{0,m,s} = 0.1$, $D_{Gth,s} = 1$ $p_{0,m,a}$ for $c_{0,m,a} = 0.3$, $D_{Gth,a} = 1$	$p_{0,m,s}$ for $c_{0,m,s} = 0.1$, $D_{Gth,s} = 1$ $p_{0,m,a}$ for $c_{0,m,a} = 0.3$, $D_{Gth,a} = 1$
Mutation schema	Insertion mutation/Deletion mutation ($p_{0,m,id}$), Lengthen/Shorten segment ($p_{0,m,ls}$), Mirror Move mutation ($p_{0,m,mm}$) Applied with equal probability of ($p_{0,m,id} = p_{0,m,ls} = p_{0,m,mm} = 1/3$ within $p_{0,m,s}$ and $p_{0,m,a}$)	Insertion mutation/Deletion mutation ($p_{0,m,id}$), Lengthen/Shorten segment ($p_{0,m,ls}$) Mirror Move mutation ($p_{0,m,mm}$), Applied with equal probability of ($p_{0,m,id} = p_{0,m,ls} = p_{0,m,mm} = 1/3$ within $p_{0,m,s}$ and $p_{0,m,a}$)
Population model	Only the best survive	Only the best survive
Starting ΔL	1/5	Not applicable
Improvement methods	Coarse Loop Removal ($p_{0,i,CLR} = 0.02$), Coarse Segments' shortening $p_{0,i,CSS} = 0.02$	Not applicable
Repair methods	Coarse Graph Repair ($p_{0,r,CCR} = 0.01$)	Not applicable
Convergence condition	All optimizing population reached their convergence condition	Best specimen fitness over the last 15 generation not decreased more than 3 %



Specimen	N_s	L_s	Violation count $ V_s = w$	$C(s)$	$f(s)$
s_0	29	3945.67	87	3319.62	87.08
s_{24}	517	62 486.61	1507	51 826.45	1507.26
s_{49}	993	119 347.96	2781	99 086.31	2783.41

Fig. 49. First candidate solutions for the Warsaw Map.

Based on experience with the simple binary map one can analyze the algorithm run for more complicated cases: for the maps with non-binary cost. In this section the case of the Warsaw Map (Fig. 48) will be analyzed. During this research the performance of a pure genetic approach will be compared with the proposed algorithm with the memetic repair and upgrade operators showing which advantage the memetic operators present over the conventional genetic ones.

A genetic algorithm differs from the proposed memetic algorithm (Table 3) by its structure and underlying properties. The main difference is the lack of a population parcellment mechanism meaning that all specimens develop within the same population. The comparison of properties of the proposed memetic algorithm and genetic algorithm is depicted in Table 5. The whole development in the genetic algorithm is done in one population and therefore one cannot compare the properties of the optimizing population of the memetic algorithm with the population of the genetic algorithm.

Therefore, we have other types of planning rules which can be violated. The importance factors have been set to $i_L = 0.03$, $i_C = 0.95$, $i_o = 0.02$ and to favor low cost path over shorter ones.

The algorithm starts with a randomly generated population (Fig. 49). Not only the obstacle encroach violations are counted here but also a hill abrupt violation and a path smoothness violation. In fact, the hill abrupt violation appears for this map appear only when an obstacle encroach violation occurs, because there is no difference in height for cost areas smaller than 1²⁶.

The first master solution appears in generation 8 and the parcellment occurs. G_0 establishes desired features of the phenotype by lowering the average number of segments. In the case of the Warsaw Map this process is approximately twice as fast (in generations) as in the case of a binary map. It is caused by counting violations twice: the obstacle encroach always occurs simultaneously with the hill abrupt and the fitness distance between non-violating and violating specimens is double than that in the case of the binary map.

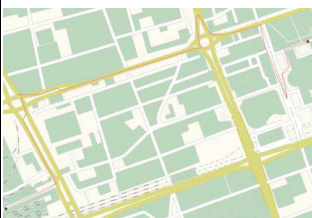
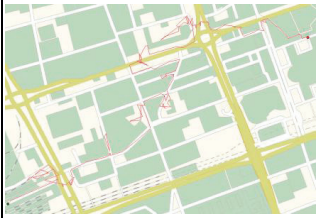
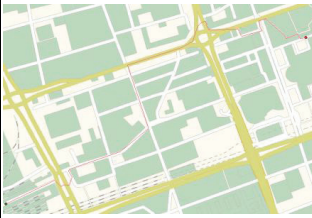
The crossover operator (both the intelligent and blind) has a desire feature of a path smoothness preservation. Therefore, if a chromosome does not violate the path smoothness planning rule, any offspring created based on such a specimen will inherit these non-violating alleles. Crossover that does not preserves the sequences of allele (e.g. the uniform crossover) will not have this feature. The obstacle encroach and hill abrupt have this useful property concerning the crossover only partially for the alleles inherited from one parent for the blind operator while the intelligent crossover minimizes the number of obstacle encroach and hill abrupt violations and produces the best possible offspring. Nevertheless, because the blind crossover dominates in G_0 , the specimens of a low fitness value can produce offspring violating the obstacle encroach and hill abrupt much more often than parents themselves but the number of path smoothness violation remains steady. On the other hand, the inheritance of path smoothness features means also that if a specimen violates this rule between some alleles, then its offspring inheriting that part of the chromosome will also violate the rule. A mechanism of offspring

²⁶ Cost equal 1 is considered as an obstacle.

selection two-on-best-one is the only one non-memetic way to improvement the path smoothness.

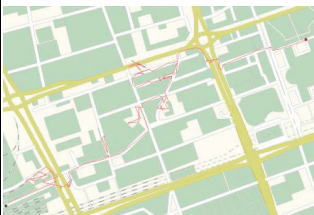
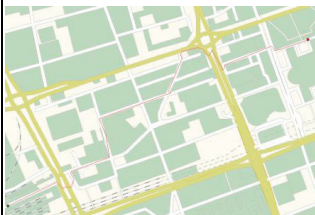
Because the biocenosis development was described in detail in the previous section, here only a summary of the biocenosis development will be given with the memetic operators employed (Table 6). For the Warsaw map the memetic algorithm converges after 175 generations producing a path of length 1358.50 and cost 268.64. The total calculation time equals 1 hour 24 minutes and 32 seconds²⁷.

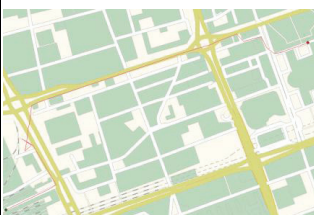
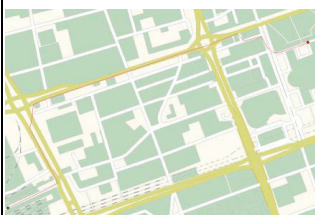
Table 6. Results of the memetic algorithm for the Warsaw map and particular generations.

Parcellment/ End generation	Master solution s_k				Population memetic operators usage		Optimizing population answer solution																
G_1 13/120					Opera- tor	No of occurrences																	
					CLR	190																	
					ELR	150																	
					CSS	295																	
					ESS	187																	
CGR	95																						
EGR	115																						
<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>792</td><td>15024.3</td><td>3196.36</td><td>0.97</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	792	15024.3	3196.36	0.97	<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>28</td><td>2121.03</td><td>293.0</td><td>0.16</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	28	2121.03	293.0	0.16
N_s	L_s	$C(s)$	$f(s)$																				
792	15024.3	3196.36	0.97																				
N_s	L_s	$C(s)$	$f(s)$																				
28	2121.03	293.0	0.16																				
G_2 19/121					Opera- tor	No of occurrences																	
					CLR	184																	
					ELR	178																	
					CSS	179																	
					ESS	159																	
CGR	100																						
EGR	102																						
<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>221</td><td>5303.24</td><td>1524</td><td>0.43</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	221	5303.24	1524	0.43	<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>21</td><td>1368.50</td><td>268.64</td><td>0.11</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	21	1368.50	268.64	0.11
N_s	L_s	$C(s)$	$f(s)$																				
221	5303.24	1524	0.43																				
N_s	L_s	$C(s)$	$f(s)$																				
21	1368.50	268.64	0.11																				
G_3 36/136					Opera- tor	No of occurrences																	
					CLR	172																	
					ELR	172																	
					CSS	169																	
					ESS	195																	
CGR	110																						
EGR	94																						
<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>202</td><td>4224.67</td><td>968.12</td><td>0.34</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	202	4224.67	968.12	0.34	<table><tr><th>N_s</th><th>L_s</th><th>$C(s)$</th><th>$f(s)$</th></tr><tr><td>34</td><td>1650.78</td><td>307.80</td><td>0.13</td></tr></table>				N_s	L_s	$C(s)$	$f(s)$	34	1650.78	307.80	0.13
N_s	L_s	$C(s)$	$f(s)$																				
202	4224.67	968.12	0.34																				
N_s	L_s	$C(s)$	$f(s)$																				
34	1650.78	307.80	0.13																				

²⁷ Most time (around 80 %) is consumed on files write/read from hard drive for data post processing.

G_4 36/160					Opera- tor	No of occurrences					
					CLR	230					
					ELR	235					
					CSS	216					
					ESS	210					
CGR	129										
EGR	115										
		N_s	L_s	$C(s)$	$f(s)$	N_s	L_s	$C(s)$	$f(s)$		
		163	4239.4	906.91	0.34			32	1921.99	359.52	0.15

G_5 65/145					Opera- tor	No of occurrences					
					CLR	130					
					ELR	125					
					CSS	143					
					ESS	144					
CGR	80										
EGR	82										
		N_s	L_s	$C(s)$	$f(s)$	N_s	L_s	$C(s)$	$f(s)$		
		159	3898.40	845.45	0.31			29	1548.25	293.68	0.12

G_6 105/175					Opera- tor	No of occurrences					
					CLR	105					
					ELR	109					
					CSS	112					
					ESS	129					
CGR	63										
EGR	73										
		N_s	L_s	$C(s)$	$f(s)$	N_s	L_s	$C(s)$	$f(s)$		
		81	1686.71	273.01	0.63			19	1530.42	220.72	0.61

The genetic algorithm with properties described in Table 5 was run. Because of the absence of memetic operators including a specialized tournament, intelligent crossover, improvement and repair methods, the first non-violating solution is found within 1134 generations. Since the algorithm tries to reduce the number of violations as in the first order, then – due to specific construction of the fitness function – it minimizes the number of segments in the population. For the Warsaw map it is impossible to find a path that violates the obstacle encroach planning rule without 15 segments²⁸ and the minimization of the number of segments below this number makes it impossible to find a non-violating path. Therefore, the probability of finding a valid path by crossover is very low and the insertion mutation plays in this case an important role because it delivers specimens with a higher number of segments. A mechanism of adapting the number of segments for the classic genetic approach is dangerous for the algorithm convergence because of one more reason. Namely, since a desired number of segments is calculated each time, the crossover

²⁸ It can be analyzed that the minimum number of „turns” that robot has to perform during movement is 14.

takes place and the average number of segments is getting lower over generations, then the diversity also diminishes putting the algorithm in local minima. Even if the population of a genetic approach consists of groups containing specimens of different local minima, it is highly unlikely that specimens of different local minima produce a high valuable offspring if mated. It is not a problem of the genetic algorithm itself but rather due to the representation type (the displacement vectors).



$N_s(s)$	w	$L_s(s)$	$C(s)$	$f(s)$
11	49	1911.62	1264.52	49.03

a) generation 0



$N_s(s)$	$L_s(s)$	$C(s)$	$f(s)$
792	15024.39	3196.36	0.95

b) generation 1134



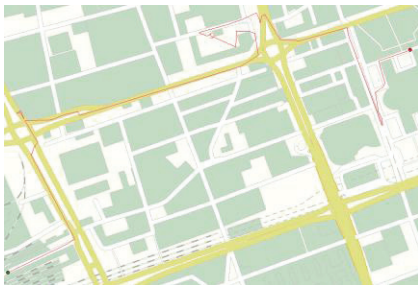
$N_s(s)$	$L_s(s)$	$C(s)$	$f(s)$
122	3664.56	729.85	0.29

c) generation 1500



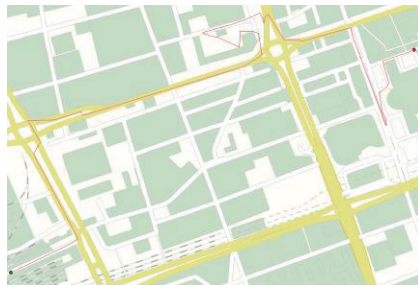
$N_s(s)$	$L_s(s)$	$C(s)$	$f(s)$
97	3036.92	588.70	0.24

d) generation 1750



$N_s(s)$	$L_s(s)$	$C(s)$	$f(s)$
55	2516.35	404.08	0.19

e) generation 2250



$N_s(s)$	$L_s(s)$	$C(s)$	$f(s)$
50	2437.16	395.70	0.18

f) generation 2400

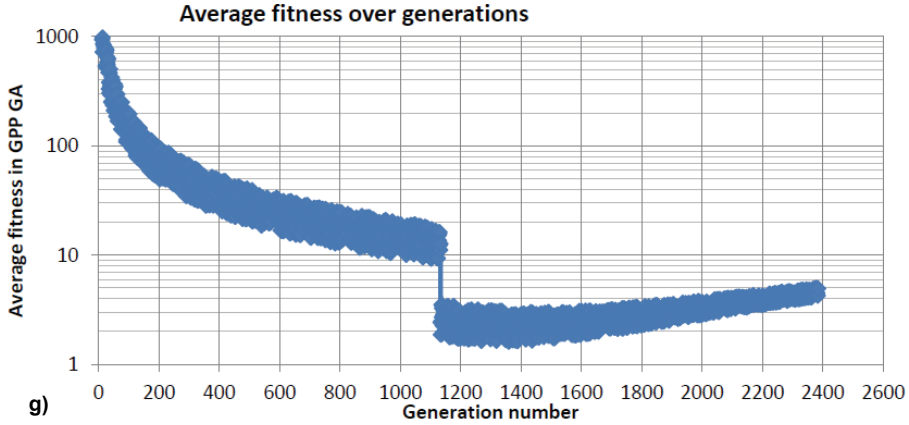


Fig. 50. The s_0 specimen over generations for the genetic algorithm; a–f for the selected generations; g) the average fitness curve for the genetic algorithm over the generations. Note a characteristic point in generation 1134 when the first non-violating specimen is found.

Apart from the representation issues, the ordinary genetic algorithm based approach will suffer from the mentioned drawbacks resulting in:

- Producing too low a number of different paths.
- Lower convergence due to lack of upgrade and repair methods and not specialized population.
- Easier convergence to local minima than in the case of more complicated maps may lead to not finding violation-free paths²⁹.

In fact, by comparing the calculation time for one generation of identical populations for the proposed memetic algorithm and the conventional genetic algorithm, the calculation time for the latter can be smaller up to 2 times. This is because of a high computational cost of the memetic operators. Clearly, this is, and will be less and less relevant due to a rapid increase of computational performance of present and future computers.

After 1134 generation the first non-violating candidate solution is found by the genetic algorithm. The average number of segments in this population is 45 and a non-violating specimen is built up of 792 segments. The path length is far away from the optimum.

Once a non-violating specimen is found, it starts to dominate over the population and the diversity starts to drop down. If during this period another unique non-violating specimen is not found, the schema of the previously found specimen will take over and the algorithm will converge to a local minimum. Analyzing the fitness plot over generations, one can see that it takes a form of a typical *COP* fitness curve. When a specimen that meets all problem constraints (planning rules) is found its schema is taking the population over and the population converges to one

²⁹ During this research the genetic algorithm was run on the HugeMaze map and did not find any violation free paths over 10000 generations.

minimum. The assenting character of the curve after 1134 generation comes from the fact that the coefficients of (43) change over generations. The faster the convergence to this local minimum the more ascending the curve becomes. It took 2400 generation for the genetic algorithm to converge which is nearly 14 times longer in the number of generations and over 9 times in the total calculation time.

To summarize the performance comparison of the genetic algorithm to the proposed memetic algorithm for the GPP problem, several facts have to be underlined:

- The proposed memetic algorithm finds several unique paths for one map. The uniqueness is defined as a distance function (52) and only as a simple comparison. Thanks to this, during the same time the calculation can be carried out fully in parallel.
- The proposed memetic algorithm does not converge to one local minimum only but rather samples the S-space and seeks for solutions laying near local minima fully in parallel.
- The convergence of each population is faster than in the case of the regular genetic algorithm thanks to the memetic operators.
- The proposed memetic algorithm produces better quality results in general because it continuously samples the S-space and the probability for attaining a near – global minimum solution is therefore higher.
- The complexity of the memetic algorithm is higher because of specialized operators. However, one should take into account that the computational time requirements do not grow by orders of magnitude and therefore an additional computational time can well be accommodated by an already high, and constantly growing performance of modern computers.
- The proposed memetic algorithm produces near optimal solutions well comparable to those obtained by the traditional visibility graph or A* algorithms.

The last numerical example is a comparison of results obtained using the proposed memetic algorithm and the well-known A* and Theta* algorithms. A* is the most commonly used algorithm for the path planning. During this test the A* algorithm searches for a path for a point holonomic robot while the memetic algorithm finds a path for the particular (nonholonomic) IBIS robot. It is an issue that the compared paths are for different robot models because of a proper decomposition cell size, and in general one cannot find path for specific platform of the same cost that for holonomic point robot.

The map is covering a region of $1002 \text{ m} \times 869 \text{ m}$ with the decomposition of 1 m^2 giving 870 738 possible configurations for the A*. This number is also the maximum number of visited cells and therefore the maximum number of algorithm iterations.

In this research the full 8-point cell neighborhood was used together with the Euclidean metrics as the basis of heuristic. This configuration is the most proper for a point robot and, combined with the Euclidean distance as a heuristic function, give best results.

During its run the memetic algorithm produced 7 optimizing populations and the last optimizing population G_7 converged in generation 221. The whole

computation process took 1 hours and 13 minutes and during the computation the memetic operators were called 889 times. The total number of populations was 8 and is limited by the algorithm itself due to its implementation limitations.

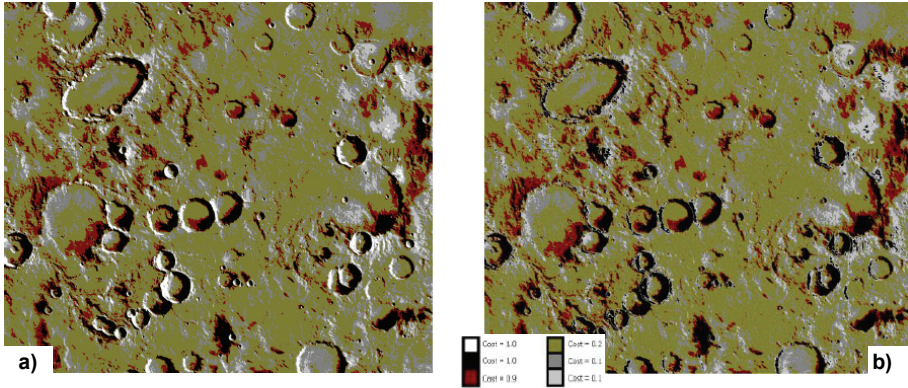


Fig. 51. The MarsMap used for a performance comparison of the proposed memetic algorithm and the A* algorithm; a) Source map b) Cost map.

A curious feature of the memetic algorithm is that the lowest cost does not mirror the best path between generations. Comparing G_7 and G_3 solutions (Table 7), one can see that even though G_7 produced a longer path of a higher cost, its best specimen s_0 has a better fitness value f_s . This fact is caused by the average phenotypic distance of specimens within the particular population. The smaller this number the higher the fitness value of the best specimen though, due to violation treatment it will not exceed 1. Therefore, a specialized population in which most specimen lay very near a local minimum will have its best specimen having the fitness of value close to 1.

Table 7. A comparison of results for The Mars map of the proposed memetic algorithm, A* and Theta*.

Generation	Start generation	End generation	Visited nodes	N_s	C_s	L_s	f_s	Calculation time [min:sec]
G_1	5	178	1 038 000	9	199.11	1014.74	0.68	45:13
G_2	17	157	2 401 023	11	243.83	9774.12	0.68	37:54
G_3	25	140	2 300 497	8	193.27	1040.16	0.69	29:50
G_4	29	190	2 898 091	55	340.17	1316.20	0.93	43:23
G_5	30	203	4 152 001	61	405.45	1336.80	0.97	50:20
G_6	31	198	2 104 249	38	282.24	1365.66	0.92	48:11
G_7	45	221	2 217 678	20	219.15	1229.47	0.61	47:20
A*	NA	NA	139 194	768	193.09	1006.11	NA	01:34
Theta*	NA	NA	2 143 526	14	226.8	975.83	NA	03:01

As it was expected, the memetic algorithm results are slightly worse than the A* results i.e. optimal cost path. The total calculation time is over 48 times longer than in the case of A* but by comparing the cost (C_s) of the path and its length L_s the results differ from each other 3.3 % in length and 0.9 % in cost. The quality of the results are therefore highly comparable. Keeping in mind that A* produces cost-optimal solutions, the proposed memetic algorithm has found a very high quality path. A* produces a sequence of neighboring cells on the decomposed maps and therefore the number of points to visit is high and equal 768. From a practical point of view, the number of intermediate path points should be minimized, and the points should be kept apart in distance. This is because each path execution is connected with the localization process and due to a position dilation, there is a change of the distance to a desired intermediate point which fluctuates in time.

Simultaneously, it changes the desired azimuth, i.e. the azimuth towards the intermediate point also changes. It is meaningless in the case of holonomic mobile robots but fluctuation of the desired azimuth, for instance for the IBIS robot, can cause an oscillatory movement which is dramatically increasing battery energy consumption. The A* algorithm calculates paths for the holonomic point robot and therefore a length related optimal path for a point robot and length related optimal path for the IBIS robot differ from each other because the planning rules differ. In general, the higher the cell decomposition resolution of the map, the more these paths will differ.

An extension of A*, the Theta* algorithm, produces on the other hand length related optimal solutions but some visibility shortcuts that it exhibits does not take into account the path cost. Therefore, a path calculated by this algorithm is much shorter, in our case 975.83 compared to 1006.11 of A* and 1040.16 of the memetic algorithm, but simultaneously the cost increases from 193.27 to 226.87, which is worse than proposed memetic algorithm. The calculation time also rises due to the visibility calculation. For the worst case, the Theta* needs to perform the visibility calculation

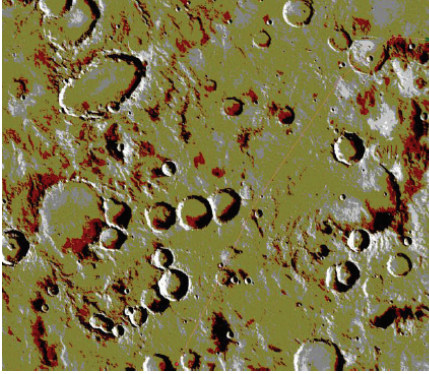
$$\sum_{i=0}^{n-1} (n-i-1)(n-i-2) \quad (55)$$

n – the number of intermediate nodes for A*.

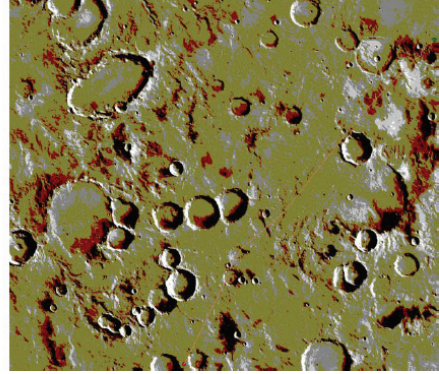
The paths produced by the memetic algorithm, and using the A* and Theta* algorithms are presented in Fig. 52.

By comparing one of the most popular GPP algorithm, A*, with the proposed memetic algorithm several facts have to be emphasized. A* and its modifications are well-known methods developed and tested on many maps. Their popularity can be explained by the fact that these algorithm are relatively easy to understand and implement. They produce complete and optimal solutions, but the optimality comes from one-criterion optimization. On the other hand, the memetic algorithm – as any other heuristic method – does not guarantee to find any solution at all but in general it performs well. The possibility of multi-criteria optimization gives it more

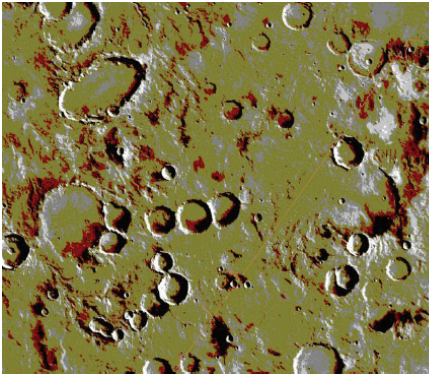
flexibility in defining priorities of a desired path but the implementation is far more complicated.



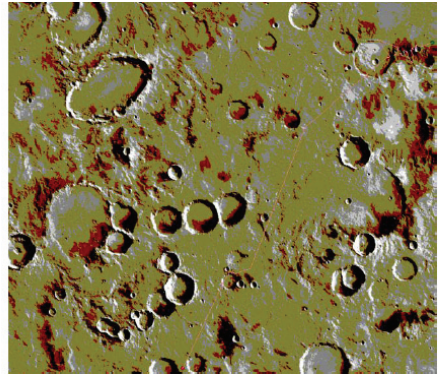
a) s_0 of G_1 , generation 178



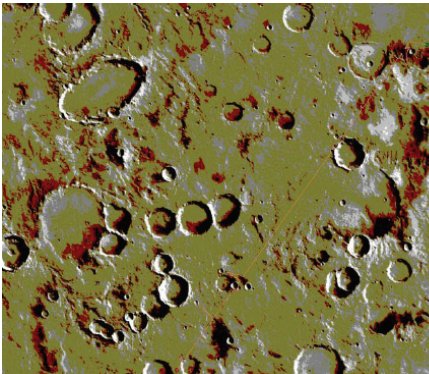
b) s_0 of G_2 , generation 157



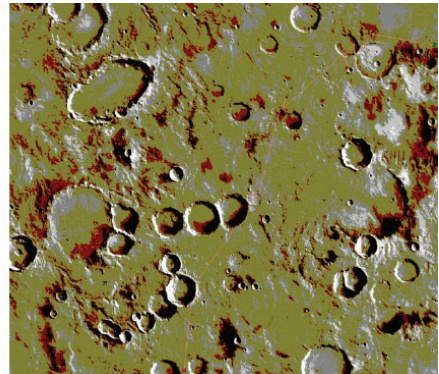
c) s_0 of G_3 , generation 140



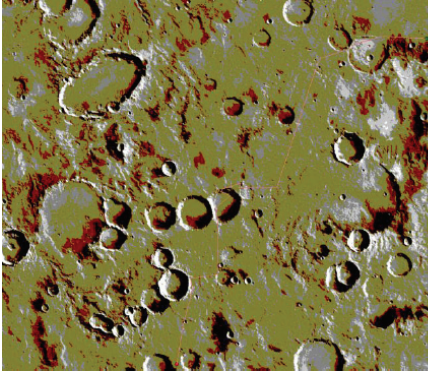
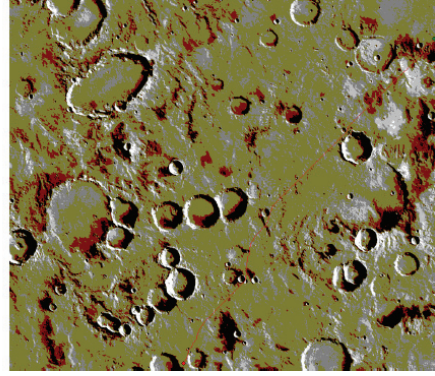
d) s_0 of G_4 , generation 190



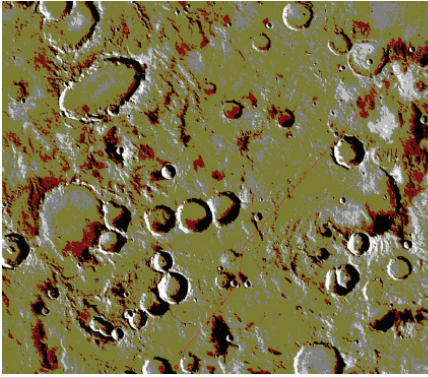
e) s_0 of G_5 , generation 203



f) s_0 of G_6 , generation 198

g) s_0 of G_7 , generation 221

h) A* solution



i) Theta* solution

Fig. 52. A comparison of solutions of the GPP for the Mars Map by using the proposed memetic algorithm a)–g) with those using A* h) and Theta* i)

A* produces paths that are not smooth and therefore another operator has to be used to make it smoother. Depending on the choice of cell neighborhood and admissible heuristic (c.f. 3.3.5) calculation, the number of visited cell can differ up to 2 times. The path is calculated for a point robot or for a circle robot when the map is dilated. More complicated robot models are difficult to implement as the transition function between cells have to be defined in advance. This would need an extension of the C-space and therefore would exponentially increase the time complexity of the algorithm – cf. Section 2.4. Main advantages and disadvantages of the proposed memetic algorithm and A* have been summarized in Table 8.

It can also be interesting to point out some other differences between A* and the memetic algorithm proposed which have been found in numerical experiments during this research. They can be summarized as follows:

- For densely cluttered environments, the performance of A* is much higher, because the unit step in this algorithm is in the neighboring eight cells. On the

- other hand, the memetic algorithm has to adapt the length of segments to the environment starting with random lengths and optimizing it over generations.
- For environments of a lower obstacle density or when intermediate points can lay far away from each other, the memetic algorithm can be computationally as fast as A*. In large maps the memetic algorithm performs better because it adapts the segments lengths to particular situations and single move (segment) iteration step far exceeds the cell iteration step.
 - Because no robot model is included in A*, there may occur situations in which an algorithm finds a path that cannot be executed (see Fig. 46). Such a drawback lead to the situation when a mobile robot needs to replant the path while getting to a dead end (the yellow circle in Fig. 46).

Table 8. A summary of advantages and disadvantages of the proposed memetic algorithm, A* and Theta* in the GPP task.

Algorithm	Advantages	Disadvantages
A*	Low complication, low complexity, fast calculations Well-known, tested Optimal and complete	Produces not smooth paths Does not take into account the robot model – the planned path is for a holonomic point robot Only serial implementation is possible No multi-criteria optimization Produces a lot of intermediate points Only one path at a time and a serial implementation is possible
Theta*	Low complication, low complexity, fast calculations Well-known, tested Optimal and complete Small number of intermediate points	Produces non-smooth paths Does not take into account the robot model – planned path is for the holonomic point robot Only serial implementation in possible No multi-criteria optimization Produces a lot of intermediate points Only one path at a time and a serial implementation is possible
Memetic algorithm	Several different paths are calculated at once Possible to include various robot models Easy definition of define path planning criteria Easy implementation of different map layers mirroring terrain features that need to be included in the GPP Possible hardware implementation for several hundred different populations in Biocenosis using the GPU Self-adapting to terrain features – possible positive behavior on dynamic maps Algorithm is complete	Complicated implementation Long calculation time Finds suboptimal solutions

4.2. Ant Colony Optimization based algorithms

Another meta-heuristic that has been used for the Global Path Planning Problem is the Ant Colony Optimization (ACO) [188]. Initially this approach was used for

finding paths through graphs [189, 190]. The idea is inspired by the way how ants explore the terrain to find food resources. Individually, each ant has poor cognitive abilities but collectively they are able to solve the problem of finding the shortest path from the nest to the food source and back.

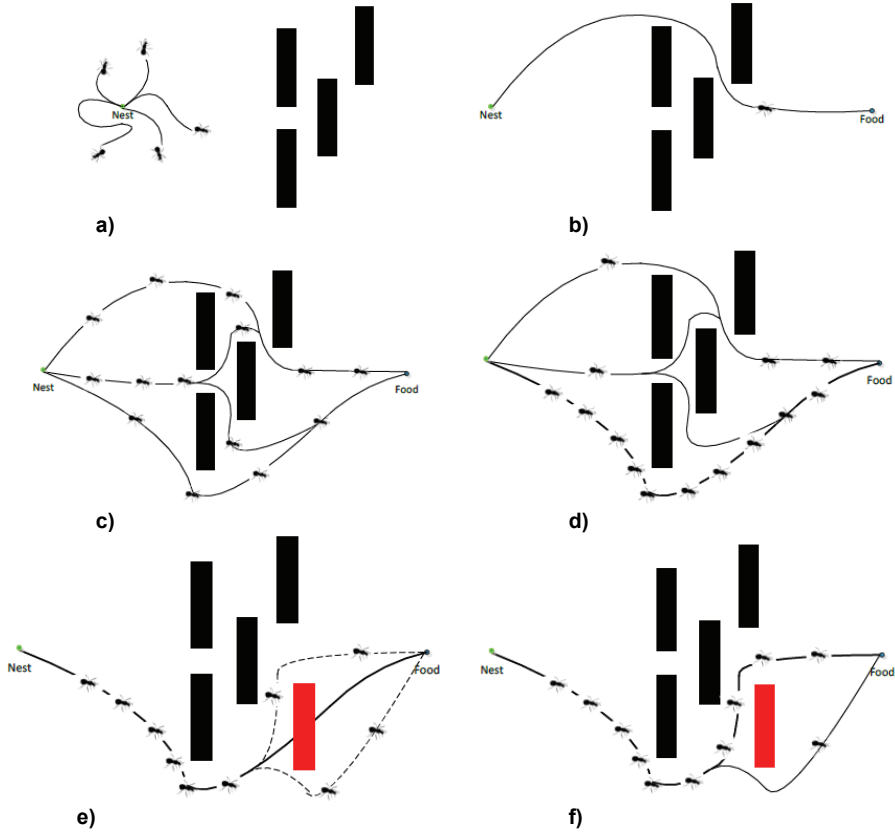


Fig. 53. An example of GPP using the ant colony optimization based method; a) at the beginning the movement is random b) the first solution is found c) another solutions are found d) the path at the bottom has the strongest pheromone (indicated by the bold pheromone line) e) a new obstacle occurs (the red rectangle) on the way and the ants loose the previous path and start to search for a way around f) new paths are found and the “best” is chosen.

Initially, all ants are walking randomly leaving vaporizing substance called a pheromone on their way (Fig. 53a). The stronger the pheromone is the higher the probability for a next ant to choose a path with that substance but over time the pheromone clearly disappears. If the ant finds a shorter route than the existing one, the pheromone attracts other ants that follow that shorter route and leave more pheromone on it (Fig. 53b). The more ants choose the new path the faster the

pheromone from the old path evaporates leading the old path to be less attractive for other ants (Fig. 53c). The process is run consecutively so that the path is updated subsequently and the algorithm reacts to dynamically changing environment (Fig. 53e and Fig. 53f).

The ant colony optimization methods works on the basis of the principle called *stigmergy*, i.e. an information between the ants is exchanged using the environment by depositing the pheromone. The convergence of the algorithm depends on several factors including the number of solutions (agents) occurring concurrently, pheromone deposit and update methods, selection of a random walk, etc. As every meta-heuristic the ant colony optimization methods were used in different variations for a wide variety of search and optimization problems including scheduling [191, 192], vehicle routing [193, 194], assignment and set tasking [195, 196], etc.

Most of approaches presented in the literature discretize the map and calculate the graph connectivity, and such a graph is searched by an Ant Colony Optimization method. Generally, in most methods [197], the map is divided into three regions: near-nest region, non-smell region, and smell region. It is assumed that food (a goal of the path plan) has a smell that attracts the ants but they can sense it in a limited area of the map. The smell area is constructed as a visibility graph from the food to any graph vertex. The near nest area is a circle around the nest of a specified radius so that there is no obstacle in this area and the ants can move freely.

For simplicity the ants move only in the directions from the nest to the food not moving back (Fig. 54b) and the probability of moving from vertex i to vertex j is equal to:

$$p_{ij}(t) \sim \begin{cases} \frac{[\tau_{ij}(t)]^\alpha}{\sum_{s \in J_k(i)} [\tau_{is}(t)]^\alpha} & \text{for } j \in J_k(i) \\ 0 & \text{otherwise} \end{cases} \quad (56)$$

$$\tau_{ij}(t+n) = \rho \tau_{ij}(t) + \sum_{k=0}^m \Delta \tau_{ij}^k \quad (57)$$

- $J_k(t)$ – directions 0, 1, 2 (Fig. 54b),
- τ_{ij} – a heuristic factor denoting the anticipated degree that ants move from grid i to j to the next grid,
- A – importance of the pheromone,
- M – number of ants,
- $\Delta \tau_{ij}^k$ – quantity of pheromone of k -th ant from i to j vertex,

$$\Delta \tau_{ij}^k = \begin{cases} \frac{1}{C^k}, & \text{if a path belongs to } T^k \\ 0, & \text{otherwise} \end{cases} \quad (58)$$

- C^k – length of the tour T^k ,
- T^k – tour made by k -th ant,
- ρ – a pheromone vaporizing factor.

The pheromone is updated according to (57), whilst the pheromone evaporates with a rate controlled by:

$$\tau_{i,j}(t+1) = (1 - \rho)\tau_{i,j}(t) \quad 0 < \rho \leq 1 \quad (59)$$

The parameter ρ is used to control evaporating rate.

Therefore, the higher the pheromone value $\eta_{ij}(t)$ is the bigger the probability $p_{ij}(t)$ of an ant to take the direction towards the cell but over time if $\sum_{k=0}^m \Delta\tau_{ij}^k$ equals 0, then no ant will follow this direction and the pheromone will start to evaporate according to the ρ factor.

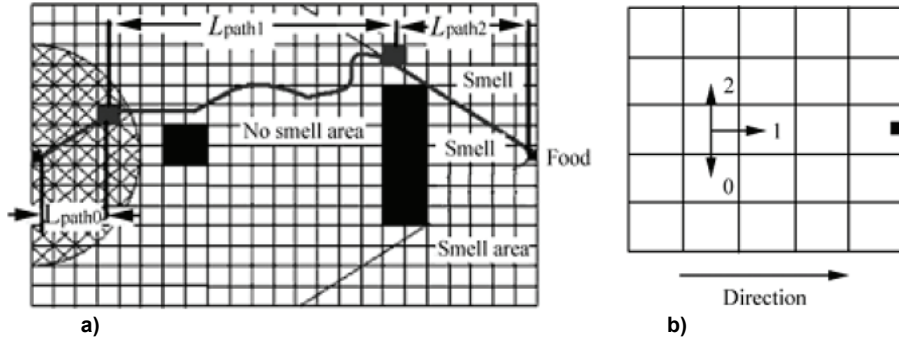


Fig. 54. Path division to L_{path0} , L_{path1} , L_{path2} .

The most popular variants of the ant colony optimization based path planning studied in the literature are:

- *Elitist ant system* [198, 199] – where the current best solution deposits its pheromone more frequent (typically at each algorithm iteration). This strengthens the best ant solution and increases the convergence.
- *Max-Min ant system* (MMAS) [200, 201] – the pheromone deposits are in the range of $[T_{min}, T_{max}]$ and only the best ant is allowed to leave the pheromone. The pheromone values are updated according to the best route length in each iteration. The initial pheromone values are set to T_{max} and the evaporation speed is limited which leads to a better exploration of the environment. If stagnation occurs [166], all pheromone values are reinitialized.
- *Rank-based ant system* [202] in which the ants deposit the pheromone according to their rank within the colony. The rank is based on the path length and only best ants, and the ants which have found a best path so far, deposit the pheromone that is updated according to:

$$\tau_{i,j}(t+1) = \tau_{i,j}(t) + \sum_{k=1}^{w-1} (w-r)\Delta\tau_{ij}^k + w\Delta\tau_{ij}^{bs} \quad (60)$$

$\Delta\tau_{ij}^{bs}$ – refers to the best path's ant pheromone deposition.

- *Continuous orthogonal ant colony* [203] – in this approach a solution is searched collaboratively. Each ant lands in the region with a probability proportional to the pheromone already deposited in order to search for a better point within the specified radius; the radius is not constant during the execution of the algorithm but is continuously updated according to a prevailing search status. After each ant finishes the search, the “desirability” of the region is reevaluated and the elitist strategy is implemented to retain the best one.

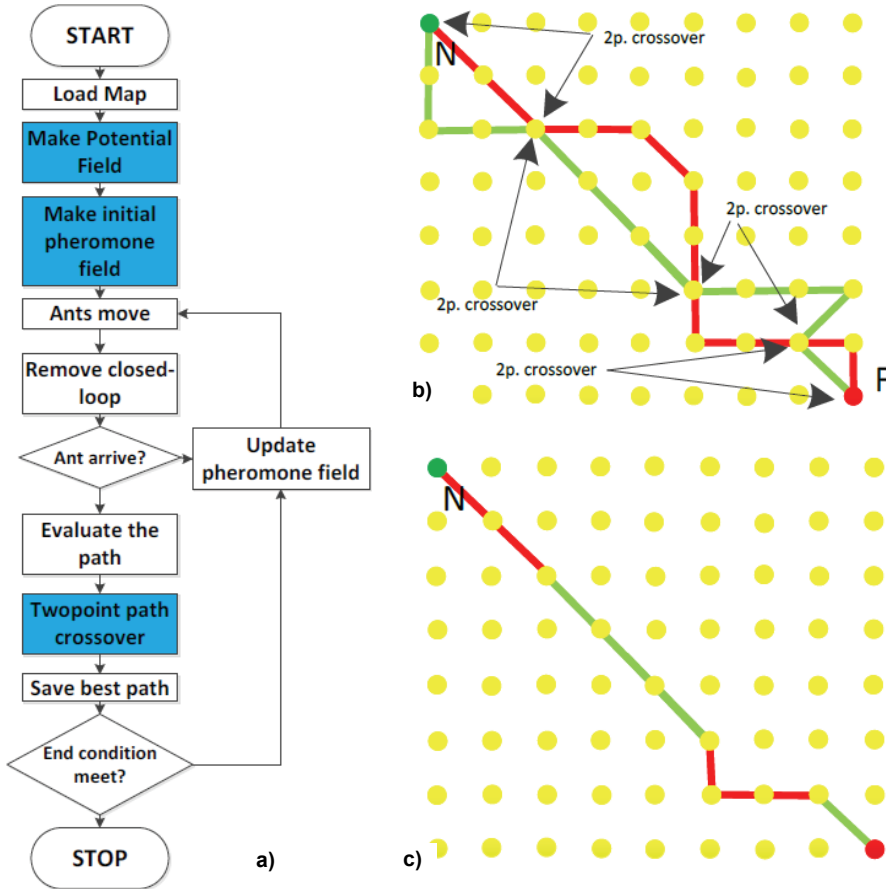


Fig. 55. Improved Ant Colony Optimization for a Global Path Planning [206]; a) The flowchart of IACO. Blue rectangles show improvements over classical ACO. b) Two point path crossover for two paths (red and green) found by ACO c) Resulting path from two point crossover operator path colors are kept constant for readability.

Number of further improvements of traditional ACO have been proposed for solving GPP problem. Most of them concern combining with the actual situation of

the robot to find a feasible region, selecting proper method of pheromone deposition, evaporation and memorizing that improves the stability of an algorithm [204]. In [205] authors exchanged a way of ant moving rule from a heuristic distance of (56) and (57) to a potential field method. Additionally the initial pheromone, not present in traditional ACO was deposited also according to potential field whilst the varying control parameter have been added to converge into the optimal solution after a preset number of iterations has been reached. Another improvement presented in [206] introduced an element called a path crossover (Fig. 55).

Since Ant Colony Optimization for a Global Path Planning methods generally suffer from slow convergence a set of methods of intensifying pheromone have been elaborated [207]. This adjustment of pheromone in already found trait depends on the quality of the path: the shorter the path is the higher the amplification of pheromone on this path is applied. In addition, if a trait has visibility point that would shorten the path such operation is performed (Fig. 56c). This approach corresponds with an improvement which Theta* is over A* algorithm (see section 3.3.5). Another fact is that as every global optimization algorithms there is always a trade-off between exploration and exploitation. Ants handle searching the path in the close proximity of the nest well, but they get hindered when approaching to the food. The pheromone accumulates over time and there is a strong tendency for new ants to follow the path that is found by a first one. To overcome this authors of [208] propose Bilateral Cooperative Exploration mechanism in addition to Modified Transition Probability Function and Modifier Pheromone Update Rate. The underlying idea is to change the goal point into the starting point and vice versa. This result in better exploration of food area. The results obtained by these operators have been gathered for a better view in Table 9 below, a similar approach of bidirectional Ant Colony Algorithms is presented in [209].

Table 9. A comparison of three Operators improving ACO for GPP which its scheme and effect on calculated path. cited after Lee et al [208].

Operator	Scheme	Effect
Modified Transition Probability Function (TPF)	Factor Normalization	Instinctive parameter selection
	Modified Visibility Field (MVF)	Reduce computation time, Considerable robot size
	Added Angle Factor	Straight-line path, Reduce computation time
Modified Pheromone Update Rule (PUR)	Unfixed Number of Ants in epoch	Fast convergence
	Limited Pheromone Quantity of each cell	Path diversity and exploration
	Path Crossover (PC)	Fast convergence Additional path without search
	Modified Pheromone Trail Value	Normalization Instinctive parameter selection
Heterogeneous Ants	Ants of Various Species	Reduce computation time Straight-line path
	Sight Expansion Principle	Straight-line path Support PC
	Interpolation of Pheromone Trail	Support PC

In order to speed-up calculations a parallel approach to ACO for the GPP have been proposed, namely a Multi Ant Colony [210]. This allows other colonies to profit from solutions found by a partial colony and also to search in different regions around the nest as pheromone matrixes are different for particular colony. Basically literature distinguish two types of multicolumn approach: heterogeneous and homogenous [211, 212]. Heterogeneous approach bases on several colonies of which ants differ in behavior: the algorithm have different set of parameters, whilst for homogenous colonies ants show similar behavior and information is exchanged every n -th generation. For a GPP problem mainly a homogenous approach is used [210] and exchanging information consists in exchange of single solution: globally best one, circular exchange of locally best solution and circular exchange of migrants. This methods correspond with co-evolutionary models such as island model [213]. In [214] authors propose multi-colony approach where colonies compete with each other. when every colony finds a trait best paths from each are compared with each other in sense of length and each colony receives a score according o their result: it is rewarded if found path is the best or penalized if not. The colony that has highest ranking keeps its pheromone traits whilst remaining colonies need to change their pheromone indexes according to an expression (61).

$$\tau_{i,j\ lo} \leftarrow \tau_{i,j\ hi} + \tau_{i,j\ lo}(1-a) \quad (61)$$

- $\tau_{i,j\ hi}$ – pheromone value for a colony of the highest ranking,
- $\tau_{i,j\ lo}$ – pheromone value for a colony of the lowest ranking,
- $a = \frac{\tau_{i,j\ lo}}{\tau_{i,j\ hi}}$ – parameter depending on the ratio of pheromone for cell (i, j) of lowest to highest ranked colonies.

Another way to improve GPP search by ACO is to use the dynamic partitioning and a heuristic function involved in angle. Expression of (56), regularly having a distance extension (63) is changed into (62) with $\eta_{ij}(t)$ of (64):

$$p_{ij}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{S \in J_k(i)} [\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}, & \text{for } j \in J_k(i) \\ 0, & \text{otherwise} \end{cases} \quad (62)$$

$$\eta_{ij}^{(1)}(t) = \frac{1}{\text{distance}(n_{i-1,j_{i-1}}, 0)} \quad (63)$$

$$\eta_{ij}^{(2)}(t) = \frac{1}{\theta_{i,j} + \varepsilon} \quad (64)$$

$$\theta_{i,j} = \arg \cos \left(\frac{\overrightarrow{n_{i-1,j_{i-1}} n_{i,j}}}{\overrightarrow{n_{i-1,j_{i-1}} n_{i+1,j_{i+1}}^*}} \right) \quad (65)$$

- ε – relatively small number to prevent $\theta_{i,j} + \varepsilon$ of (64) not equal to 0,
- $n_{i-1,j_{i-1}}$ – ants current location node,
- $n_{i+1,j_{i+1}}^*$ – optimal node at the $(i+1)$ -th node on the line.

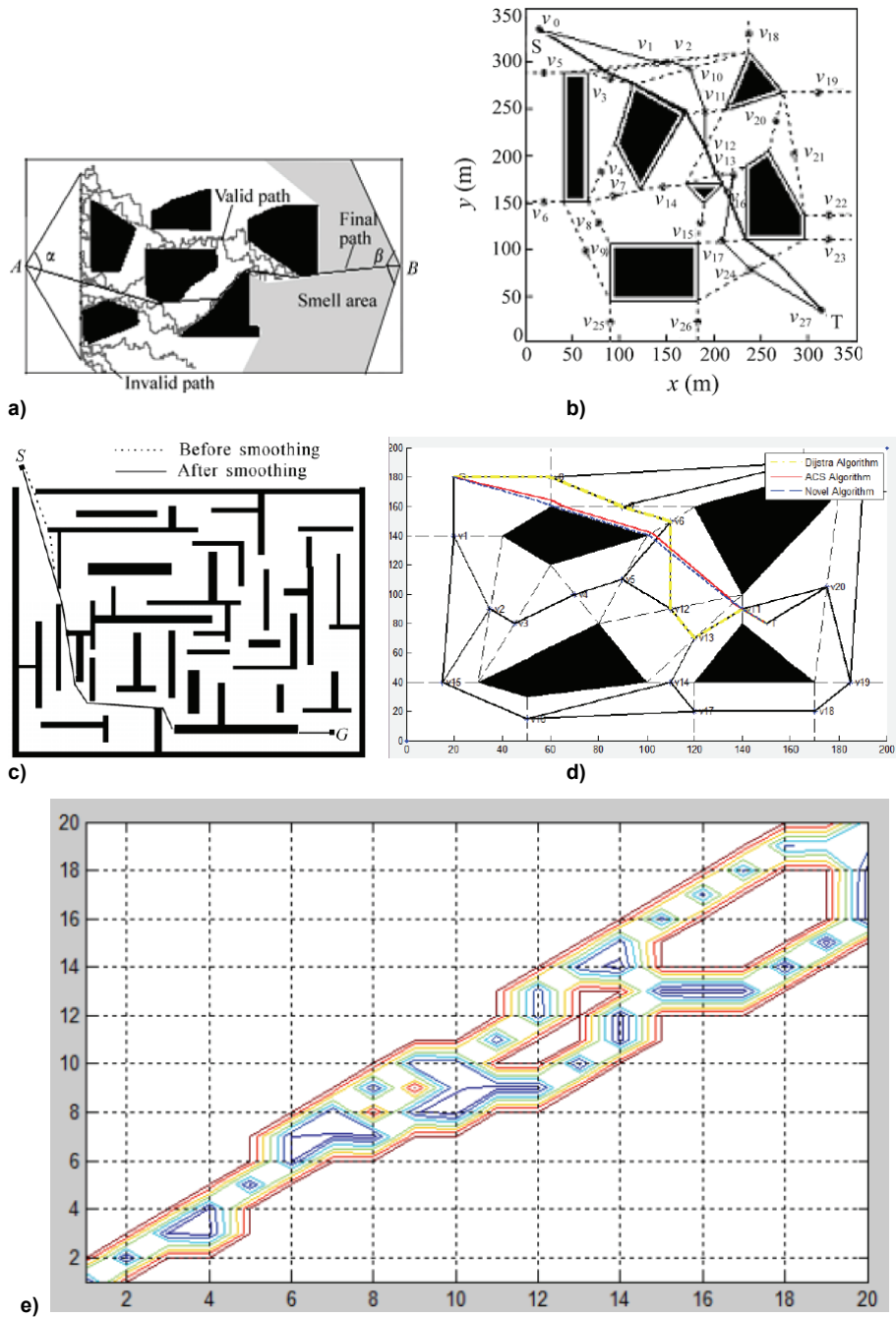


Fig. 56. Some examples of path planning using the ant colony optimization. Note that the final paths of algorithms are close to the results obtained by using the visibility diagrams: a [197], b [198], c [214], d [217], e[214].

Therefore the smaller the angle (64) to be the higher probability that this path is chosen. Therefore paths towards the goal are preferred and searched first of all.

Garro [215, 216] suggests to use the evolving ant colony system *ACO-GA* where the number of ants and other parameters of the colony change over time due to a genetic algorithm.

Examples of results obtained by the ant colony optimization path planning are shown in Fig. 56.

4.3. Particle swarm based optimization GPP

A particle swarm optimization introduced by Kennedy and Eberhart [218] is another metaheuristic widely employed to solve GPP. It is a population based approach for which an algorithm operates on a group of candidate solution. Each solution is called a particle whilst a population of solutions is a swarm. During algorithm run particles move their position on a search space remembering their own best position and the best position of a swarm. The position in the next iteration of an algorithm depends on these two. Basically two parameters controls a next position of a particle: its current position and the velocity which depends on best particle's position and best position in the swarm. A pseudo code for this method is shown in figure below.

```

I. For each particle  $i = 1, \dots, S$  do:
  1. Initialize the particle's position with a uniformly
     distributed random vector:  $\mathbf{x}_i \sim U(\mathbf{blo}, \mathbf{bup})$ , where  $\mathbf{blo}$ 
     and  $\mathbf{bup}$  are the lower and upper boundaries of the
     search-space.
  2. Initialize the particle's best known position to its
     initial position:  $\mathbf{p}_i \leftarrow \mathbf{x}_i$ 
  3. If ( $f(\mathbf{p}_i) < f(\mathbf{g})$ ) update the swarm's best known
     position:  $\mathbf{g} \leftarrow \mathbf{p}_i$ 
  4. Initialize the particle's velocity:
      $\mathbf{v}_i \sim U(-|\mathbf{bup}-\mathbf{blo}|, |\mathbf{bup}-\mathbf{blo}|)$ 

II. Until a termination criterion is met (e.g. number of
iterations performed, or a solution with adequate objective
function value is found), repeat:
  1. For each particle  $i = 1, \dots, S$  do:
    - For each dimension  $d = 1, \dots, n$  do:
      • Pick random numbers:  $r_p, r_g \sim U(0,1)$ 
      • Update the particle's velocity:
         $\mathbf{v}_{i,d} \leftarrow \omega \mathbf{v}_{i,d} + \phi_p r_p (\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g (\mathbf{g}_d - \mathbf{x}_{i,d})$ 
      • Update the particle's position:
         $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$ 
  2. If ( $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ ) do:
    - Update the particle's best known position:  $\mathbf{p}_i \leftarrow \mathbf{x}_i$ 
    - If ( $f(\mathbf{p}_i) < f(\mathbf{g})$ ) update the swarm's best known
      position:  $\mathbf{g} \leftarrow \mathbf{p}_i$ 

```

Fig. 57. A pseudo code for simple Particle Swarm Optimization

PSO have as for instance genetic algorithm both type of operators: unary and ternary. An Unary operator changes the velocity of the particle based on its best position, while ternary is included in velocity change based on swarm best position which makes it a useful utility in path planning.

PSO is often used as an optimization method rather than a search one. In [219] path planning algorithm finds Ferguson Splines [220] and considered path planning algorithm is equivalent to optimizing parameters of Ferguson Splines. Basically spline connecting point P_0 and P_1 with the tangent vectors $\vec{P}_0$ and $\vec{P}_1$ is described as following [221]:

$$r(t) = P_0 a_0(t) + P_1 a_1(t) + \vec{P}_0 a_2(t) + \vec{P}_1 a_3(t) \quad (66)$$

$$\begin{cases} a_0(t) = 2t^3 - 3t^2 + 1 \\ a_1(t) = -2t^3 + 3t^2 \\ a_2(t) = t(t-1)^2 \\ a_3(t) = t^2(t-1) \end{cases} \quad (67)$$

It is therefore a polynomial spline. Vectors $\vec{P}_0$ and $\vec{P}_1$ are calculated from substitution of derivation of (67). A path in [220] is a vector of consecutive splines: first spline's end point is equal to second spline's beginning point and a ending tangent vector of first spline is equal first tangent vector of second spline to maintain path smoothness. A particle is coded as spline parameters. Fitness function penalize long paths but most of all it should increase fitness value for splines that encroach obstacles:

$$f(s) = \frac{l(s)}{l_{min}} + \alpha f_2(s) \quad (68)$$

$$f_2(s) = \begin{cases} 1 & \text{for } d_{min} > D_{safe} \\ \frac{D_{safe}}{e^{d_{min}+1}} & \text{for } 0 < d_{min} < D_{safe} \end{cases} \quad (69)$$

- D_{safe} – minimum distance of the robot to approach an obstacle (constant),
- d_{min} – minimum distance of particular path to an obstacle,
- α – weighting factor to adjust the proportion of the length and the safety.

Similar way of expressing fitness is to evaluate each position by taking into an account its distance to the goal position and distance from nearest obstacle [222]:

$$f(p) = w_1 \frac{1}{\min_j \|p - o_j\|} + w_2 \|p - T\| \quad (70)$$

- $\|\cdot\|$ – distance function,
- p – position of robot,

- o_j – j -th obstacle,
- T – target position,
- w_1 – obstacle clearance scaling factor,
- w_2 – distance to goal position scaling factor.

Such definition allows implementation of the method for environments with moving obstacles if only obstacle description method would be efficient enough so that path planning method is able to assess the position of moving obstacle (and in result a distance to obstacle $\min_j \|p - o_j\|$ (75)) properly in order to incorporate its position into calculations. Another way is to perform an optimization of graph path on the MAKLINK graph [223–225] or a visibility graph [226].

Two new methods are introduced as an extension of classical PSO in [227]. These are the chaotic particle swarm optimization and second order oscillating PSO. Both of these methods modify the cognitive and social behaviors, i.e. the velocities dependence on own and best particle position. In the chaotic particle swarm optimization the novel approach lies in the initialization of velocity and position of a single particle as well as the treatment of global best position. Most of particle positions are generated by so called chaotic sequence [228]. This keeps the randomness of initial population. For the second order oscillating PSO a velocity of classical PSO of (76) of a particle is calculated as follows (72) [227]:

$$v_i(t+1) = wv_i(t) + \varphi_1(p_i - x_i(t)) + \varphi_2(p_g - x_i(t)) \quad (71)$$

$$v_i(t+1) = wv_i(t) + \varphi_1(p_i - (1 + \xi_1)x_i(t) - \xi_1x_i(t-1)) + \varphi_2(p_g - (1 + \xi_2)x_i(t) - \xi_2x_i(t-1)) \quad (72)$$

- $v_i(t+1)$ – velocity of i -th particle in the next iteration,
- p_i – best particle's solution,
- p_g – best population's solution,
- $x_i(t)$ – particle,
- φ_1, φ_2 – cognitive and social scaling factors,
- ξ_1, ξ_2 – oscillation coefficients for cognitive and social behavior.

The velocity is therefore calculated based on previous and one before previous velocities values reaching better convergence.

Further extensions of PSO aim in specialization of a search i.e. hybridizing an algorithm by incorporating problem-specific knowledge. In [225] particles are searched within a narrowed region of a search space called a “active region” which reduces a number of probable particle configuration that are not crucial for a global path planning problem. In [229] authors propose to replace invalid paths (particles) with global best solution or local best solution rather than reinitialize particle.

Further extensions of PSO aim in specialization of a search i.e. hybridizing an algorithm by incorporating problem-specific knowledge. In [225] particles are searched within a narrowed region of a search space called a “active region” which reduces a number of probable particle configuration that are not crucial for a global path planning problem. Also the mechanism of replacing invalid candidate solutions – particles global solution or local optima reduces iteration time by removing the necessity of reinitialising particles.

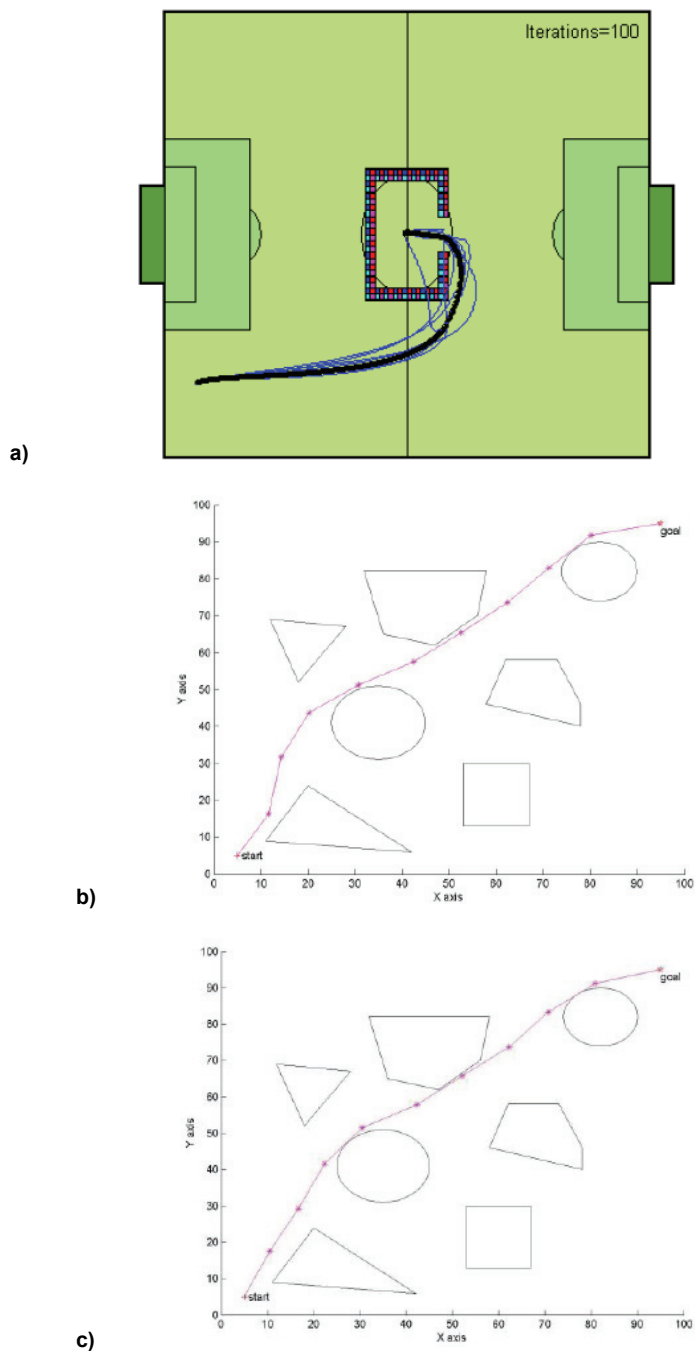


Fig. 58. Some examples of path planning using the particle swarm optimization; a) Ferguson splines PSO [219]; b) Chaos PSO [227]; c) Second Order PSO [227].

In [229] authors propose to replace invalid paths (particles) with global best solution or local best solution rather than reinitialize particle. This avoids premature convergence by harmonizing global search and local search abilities whilst gaining rapid convergence to the satisfactory solution. The authors achieve the goal by applying static, dynamic, annealing and adaptive penalty functions. Some examples of global path planning by particle swarm optimization can be found in Fig. 58.

4.4. Simulated annealing in GPP

Simulated Annealing is a simple metaheuristic generic algorithm for a global optimization with a given function. It is widely used when the search space is discrete – as in the case of GPP. It was introduced by Kirkpatrick, Gelatt, and Vecchi [230] and is an adaptation of Metropolis–Hastings algorithm [231] which is a Markov chain Monte Carlo method [232]. The name comes from an inspiration of annealing in metallurgy that needs a specific procedure for heating and cooling.

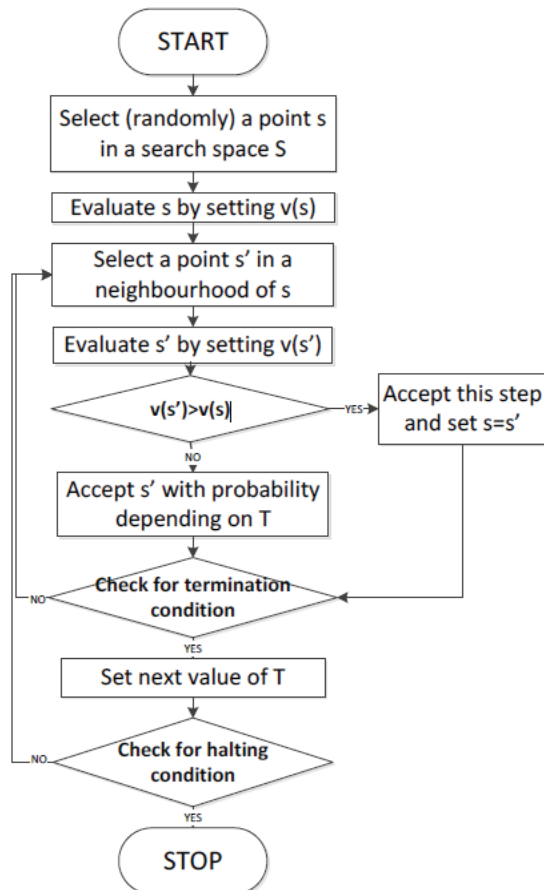


Fig. 59. The outline of simulated annealing. Maximization problem is assumed.

The algorithm consists in iterative sampling of a search space for solution (Fig. 59). It starts from selecting a point s in a search space. Next a sampling for a next possible step s' in the neighbourhood of s is performed. If s' has a better assessment than previous point s , it is unconditionally accepted ($s \leftarrow s'$) and if the value of $v(s')$ does not satisfy termination condition the process is repeated. On the contrary if $v(s') < v(s)$ it is accepted with a probability p , that depends on parameter (temperature of cooling) T :

$$p = e^{\frac{v(s') - v(s)}{T}} \quad (73)$$

Once the termination condition is met for current value of T the next (smaller) T is set according to cooling ratio $g(T, t)$ and whole process starts from the very beginning until halting criterion. The temperature T controls the “acceptance ratio” for worse than current solution, whilst cooling ratio $g(T, t)$ sets the way how this acceptance changes over time.

To summarize: in the simplest case a simulated annealing needs to define following elements prior to run: the neighborhood function of current point in a search space, initial temperature T_{init} , the cooling ratio $g(T, t)$ and the minimum temperature T_{min} which determines halting criteria. T_{init} , $g(T, t)$ and T_{min} control the algorithm’s behavior whilst the neighborhood search is purely problem related – GPP path search will focus mainly on this operator.

A classic approach for a path planning using Simulated Annealing is presented in [233]. The evaluation criterion is the path length. The map consists in convex polygons, while a path is represented as a sequence of vertices. Therefore algorithm of path planning considered is a graph solving technique. The pseudo code for this technique is presented below in Fig. 60. Obstacles are dilated for assuring robot’s minimum distance to obstacles (dashed line in Fig. 60b) and vertices V of each polygon are determinate (yellow dots in Fig. 60b). After that a visibility of each vertices pair is defined.

Vertices sampling starts from B node and the inner loop of an algorithm collects following vertices on the path. In Fig. 60b) a result of a run of an inner loop produced a path: $[B, V_{16}, V_6, V_2, V_{11}, V_{10}, V_{14}, V_9, V_{19}, E]$. However presented method is optimal in sense of path length³⁰ the last step of an algorithm (after cooling is done) would produce a path identical to visibility diagram (see section 3.1.1), whilst sampling technique will be identical to A* star method (see section 3.3.5).

Several improvements over classical Simulated Annealing have been proposed for a Global Path Planning. In [234] authors propose best reserve mechanism. It consist of remembering the best solution of a search process meeting in a so-called conservator. Results from pure SA and conservator are compared after halting condition and a superior one is chosen. Another way is to employ Improved simulated annealing mixed optimize algorithm. This uses a Powell [235] approach for optimizing multi-dimensional functions (as a fitness function of GPP is). The search process is done successively $n + 1$ times. For each of these steps a lineary independent direction is chosen. If best solution on that direction is found another

³⁰ If only one considered obstacle dilation performs as a preparatory step of an algorithm.

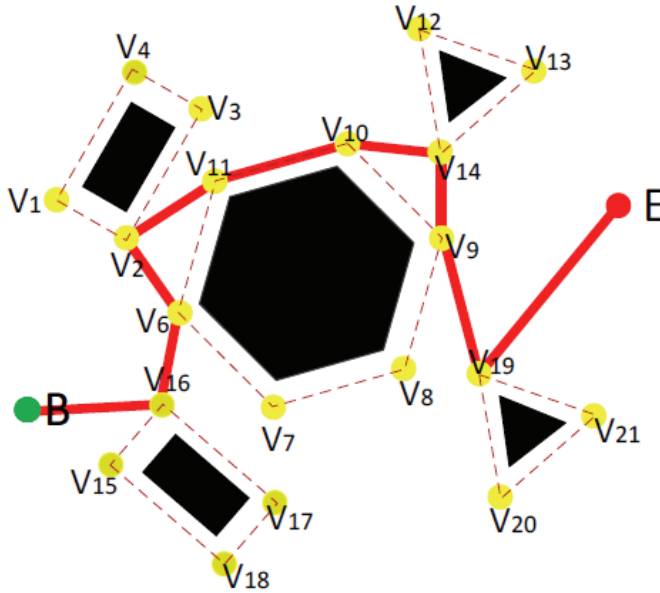
step is performed: new linearly independent direction towards the starting point until all n directions are penetrated. After that one of n -th direction is replaced by $(n + 1)$ -th and then the algorithm begins in new phase.

a)

```

T = Tinitial;
while (T > Tterminate)
    randomly generate one feasible solution  $X_s$ ;
    evaluate  $X_s$ , count = 1;
    while (count < Threshold)
        generate a new feasible solution  $X_n$  base on  $X_s$ ;
        evaluate  $X_n$ ,
        if  $f(X_n) < f(X_s)$   $X_s = X_n$ ;
        else if  $\text{rand}(1) < (\exp((f(X_s) - f(X_n))/T))$   $X_s = X_n$ ;
        count = count + 1;
    endif
endwhile
T = cool rate · T;
update  $X_s$  at each reduction of temperature T
endwhile
 $X_s$  is the optimal or near-optimal solution for the robot.

```



b)

Fig. 60. a) The pseudo code for Simulated Annealing for GPP; b) Example path by simulated annealing method presented in [233].

Basically Simulated Annealing itself does not employ a clear definition of a neighborhood search. Therefore in most cases it is combined with other methods. In [236] an integration of SA with artificial potential field is proposed. Artificial

Potential field is basically a steepest descent method and it is greedy in its behavior, whilst simulated annealing is an iterative stochastic “hill climber”. Therefore connecting these methods is natural such as iterative greedy algorithms developed into methods like simulated annealing or taboo search.

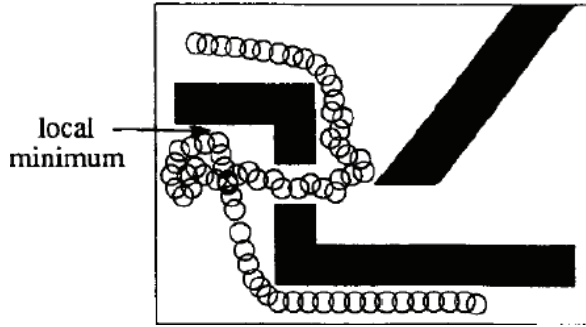
The initial path for this methods consists in a list of nodes similar to the one obtained from visibility graph. In order to increase the robot-obstacles distance are subjected to the potential function describe by (74). Additionally two other forces are employed: U_{length} of (75) and U_{cross} of (76) for keeping the path shortest possible and to maintain small number of obstacle crossings respectively.

$$U_{cen(i)} = K_{cen(i)} \left(1 + \frac{1}{\rho_{cen(i)}} \right) \quad (74)$$

$$U_{length} = K_{length} L \quad (75)$$

$$U_{cross} = K_{cross} N_{cross} \quad (76)$$

$K_{cen(i)}$, K_{length} , K_{cross} – scaling factors,
 $\rho_{cen(i)}$ – distance to the centroid of obstacle i ,
 L – length of the path,
 N_{cross} – number of crossing to the obstacle boundaries.



```

Set Path = Start-path (for simulated annealing),
Set T = T0,
While T ≥ Tj and not escaped, perform this loop:
    Pick random point P out of the points in Path that indicate
    a collision with the obstacle.
    Set Path' to random neighbour path of Path (which
    neighbours Path at P).
    Set  $\Delta = U(\mathbf{Path}') - U(\mathbf{Path})$ .
    If  $\Delta \leq 0$ , set Path = Path'.
    If  $\Delta > 0$ , set Path = Path' with probability  $e^{-\Delta/T}$ .
    If  $U(\mathbf{Path}') < U(\mathbf{Start-path})$ , then successful escape.
    Set T = rT.
If not escaped then return Failure, else return Path.

```

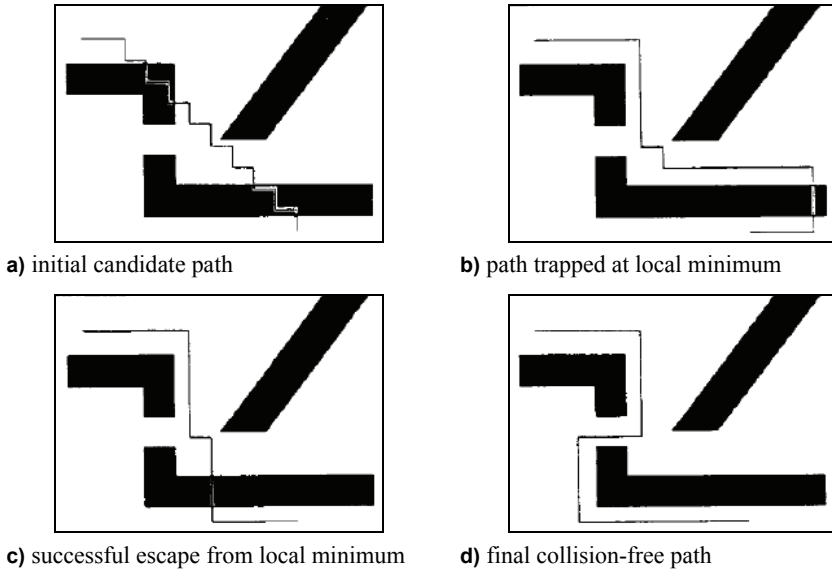


Fig. 61. a) Sample minimum of the field to escape from by Simulated annealing method; b) Algorithm pseudo code; c) Results obtained by introducing additional potential functions.

Basically a APF is calculated on the map and Simulated Annealing is used to escape local minima of the field (Fig. 61a). Therefore SA is used to sample next configuration in the field. Global Path Planning tries to minimize the entire path potential value by exchanging these parts of the path that are of high potential to ones that have smaller potential. In [236] it is done by introducing additional potential functions for algorithm with block schema is presented in Fig. 61b.

Similar approaches can be found in many papers, from which the most famous are [237, 238]. Remaining methods that make use of Simulated Annealing presented amongst all in papers such as [234, 239, 240] base on Artificial Neural Networks.

5. Summary

The *GPP* problem is one of the most crucial and important problems of robotics and is one of two foundations of movement autonomy of any unmanned ground vehicle, including mobile robots. Over 40 years of research have resulted in the development of a large group of optimal and heuristic methods and we still witness an active research activity to develop new methods mainly because of an unsatisfactory computational efficiency of the known methods, their inflexibility, their quite limiting underlying assumptions, a difficulty with their combination with the *LMP* methods, etc. Under the most common assumptions the *GPP* problem is *NP*-hard [9] – for a full review of computational complexity we refer the reader to Chapter 3 of [241]. In general, the methods developed are evaluated with respect to time and memory space complexity, completeness and optimality. A path planner is said to be *complete* if it always returns a path, if one exists, but also informs if a path does not exist. Moreover, a *GPP* algorithm is called *optimal* if the returned path is optimal under the performance function and assumptions adopted.

All the methods for solving the *GPP* problem can be divided into several categories:

- grid and decomposition based algorithms,
- computational geometry algorithms,
- potential fields algorithms, sampling based algorithms, and
- meta-heuristics based algorithms,

but all of them transform the real world into the *C*-space that has to take into account geometric features of the robot. Choosing a proper *C*-space representation is a key point because it implies the use of one or more methods listed above.

The earliest approaches [44, 66, 109] exemplified by those based on the Voronoi/visibility diagram were based on geometrical computations and constituted simple deterministic methods of building a roadmap. Moreover, they represented methods with a point robot (holonomic) and polygonal obstacles where roadmap between points were calculated. Comparing them to the currently used methods it is easy to see that they were not computationally complex, and also were characterized by the fact that just one criterion was optimized at a time, exemplified by the maximization of the robot-to-obstacle distance in the case of the Voronoi diagram, and the path length in the case of the visibility graph.

In the potential field approach [30, 94–96], the robot is seen as a particle in the artificial field that consist of two fields:

- a goal attracting the robot, and
- an obstacles repelling it.

The never ending problem with local minima is serious here. In the group of graph algorithms, the most common technique consists in creating the topological connections between points in mostly regular grid and then calculating a graph path from the start to goal position. In many cases the path produced by the graph is

suboptimal. As the number of graph vertices grows, the planned path becomes smoother but the computational time increases dramatically.

The sampling methods are another group of possible solutions to the *GPP* problem. The C-space is sampled there and a graph is created. Regular sampling leads to grid-based methods but probabilistic methods sample the C-space on the basis of different circumstances such as the presence and setting of obstacles. The configurations sampled form then a graph which is then solved for finding a path.

As the methods mentioned were have been considered to be inflexible, the use of various meta-heuristics has started to dominate over the last 20 years. For these methods, in general, time and space complexity cannot be determined and in theory they cannot guarantee to yield results at all. Nevertheless, in practice the results they yield are near-optimal. Also, though the convergence of these methods is not guaranteed, since each iteration produces better and better results than the previous one, they guarantee a constant progress. On the other hand, the great advantage of them is easiness to incorporate multiple objectives in the optimization so that not only the shortest paths is being found but also a path that has other desired properties exemplified by the path cost. The use of genetic algorithms yields several candidate answer during one iteration of the algorithm and is insensitive to dynamic environment changes.

For an easy comparison of the GPP algorithms, their main characteristics are summarized in Table 10.

Table 10. A summarization of Global Path Planning Algorithms including main properties: optimality, completeness advantages and disadvantages.

Method name	Year	Main contributors	C-space representation	Complexity	Completeness
Visibility graph	1969	Lozano-Pérez, Wesley, Nilsson	Two dimensions, but the idea of visibility can be extended to any number of degrees in C-space	$O(n \log n + E)$	YES
Triangulation methods	1984	Lee, Preparata	Two dimensions	$O(n)$	YES
Voronoi diagrams	1644	Shamos	Two (three) dimensions	$O(n \log n)$	YES
Probabilistic roadmaps	1996	Kavraki	No-limitation	Cannot be determined ³¹	Probabilistically
Potential fields	1986	Khatib	Three dimensions (position only)	Cannot be determined	NO
Wavefront planners	1991	Latombe, Barraquand, Brock	Up to four dimensions	$O(n)$	YES
Depth-first search Breadth-first search			Any number, but the traversal cost has to be specified	$O(V + E)$	YES
Uniform cost search and the Dijkstra algorithm	1959	Dijkstra	Any number, but the traversal cost has to be specified	$O(b^c/M)$	YES
Trulla algorithm	1991	Mitchell	Any number, but the traversal cost has to be specified	Cannot be determined	YES
A* algorithm and variations	1968	Hart, Nilsson	Any number, but the traversal cost and heuristics function h have to be specified	At least $O(\log h^*(x))$	YES
Simulated Annealing	1983	Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P.	Any number	Cannot be determined	NO

³¹ The basis of the method is probabilistic.

Path optimality	Advantages	Disadvantages
YES in the sense of path length	Simple to calculate	Polygonal convex obstacles only, single criteria optimization, works only for binary maps, robot treated as point, mainly for static environments
NO, if a path relaxation technique is used the method give identical result to visibility graph	Low computational complexity, a map does not have to be of a rectangloid shape	Polygonal convex obstacles only, single criterion optimization, works only for binary maps, robot treated as a point, requires a path relaxation, mainly for static environments
YES (in the sense of maximization of the robot-to-obstacle distance)	Easy to calculate by the divide and conquer method, it is easy to incorporate disc shaped robots	Non-optimal in the sense of distance, no multi-objective optimization, only for three dimensional C-space, mainly for static environments
NO – only near an optimal solutions	A lot of different sampling techniques, possible for non-holonomic robots, easiness in defining distances between non-position configurations	Computationally expensive, each iteration takes more time, the paths produced are jagged – a smoothing technique required, mainly for static environments
NO	Easy and intuitive to calculate, robust, produces smooth paths	Local minima that need additional techniques to overcome, hard to incorporate more dimensions in the C-space, mainly for static environments, needs first the cell decomposition
YES	Simple to implement, may include traversal cost	Needs smoothing, produces paths that are closely hugging obstacles, need a wall following behavior for support, mainly for static environments
YES	Simple to implement	Non-optimal search that depends on the branching factor or graph depth, only for non-negative edges weights
YES	Simple to implement, produces paths for each vertex	Incorporates edge weights, only for non-negative edges weights
NO	Easy to combine with LMP	Needs prior decomposition, produces paths similar to the wavefront planners
YES	Low time complexity, easy to implement	Path produced is not smooth, can be used mainly for the up to 3-dimensional C-space because it is hard to define a heuristic function; the number of path subgoals can be decreased by using the Theta* extensions
NO	Easy to implement	Need a special tuning (no self tuning mechanism). A generic method so that the S-space and C-space need to be prepare using other methods prior to path planning

5. Summary

Method name	Year	Main contributors	C-space representation	Complexity	Completeness
Genetic Algorithms	1991	Xiao, Michalewicz	Any number	Cannot be determined	NO
Ant Colony methods	1991	Dorigo, Hoos, Stützle	Any number, but a specific pheromone deposit function has to be elaborated	Cannot be determined	NO
Particle Swarm Optimization	1995	Kennedy, J.; Eberhart, R	Two dimensions, but often S-space is transformed	Cannot be determined	NO

Path optimality	Advantages	Disadvantages
NO	Several paths are calculated in parallel; most of the genetic algorithm operators can be calculated in parallel multiple criteria possible; can be applied to dynamic environments	Complicated to implement; uses more computer resources, notably for offline planning
NO	Several near optimal path are calculated; can be easily implemented for dynamic environments	Computationally expensive; newer methods are difficult to implement
NO	Easy to implement, more efficient than exhaustive search. Population based search. Small number of parameters to tune	Poor convergence. Only one parameter for controlling exploration and one for exploitation, therefore these tasks cannot be reconcile

6. References

1. Y. K. Hwang, N. Ahuja, "Gross motion planning – a survey", *ACM Computing Surveys*, vol. 24, pp. 219–291, 1992.
2. M. A. Batalin, G. S. Sukhatme, M. Hattig, "Mobile robot navigation using a sensor network", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 636–642, 2004.
3. I. Ulrich, J. Borenstein, "VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1572–1577, 1998.
4. A.S. Miralles, M.A.S. Bobi, "Global Path Planning in Gaussian Probabilistic Maps", *Journal of Intelligent and Robotics Systems*, vol. 40, pp. 89–102, 2004.
5. J. Giesbrecht, *Global Path Planning for Unmanned Ground Vehicles*, Defence R&D Canada – Suffield, 2004.
6. M. Gao, J. Xu, J. Tian, "Mobile Robot Global Path Planning Based on Improved Augment Ant Colony Algorithm", *Proceedings of 2nd International Conference on Genetic and Evolutionary Computation*, 2008, pp. 273–276.
7. R. Siegwart, I. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, 2004.
8. J. Song, B. Dai, H Cui, E. Shan, H. He, "An Adaptive Rolling Path Planning Method for Planet Rover in Uncertain Environment", *Intelligent Robotics and Applications*, Springer 2009, pp. 1023–1031.
9. S.M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.
10. T. Ersson, X. Hu, "Path Planning and Navigation of Mobile Robots in Unknown Environments", *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, vol. 2, pp. 858–864, 2001.
11. J. Barraquand, J.-C. Latombe, "Robot Motion Planning: A Distributed Representation Approach", *The International Journal of Robotics Research*, vol. 10, no 6, pp. 628–649, 1991.
12. J.-D. Boissonnat, F. Avnaim, B. Faverjon, "A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles", *Proceedings of 5th IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1656–1661, 1988.
13. K. Kant, S.W. Zucker, "Toward efficient trajectory planning: the path-velocity decomposition", *The International Journal of Robotics Research*, vol. 5, no 3, pp. 72–89, 1986.
14. G. Dudek, M. Jenkin, *Computational Principles of Mobile Robotics*, Cambridge University Press, 2000.
15. S. Quinlan, O. Khatib, "Elastic bands. Connecting path planning and control", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, pp. 802–807, 1993.

16. R. Manduchi, A. Castano, A. Talukder, L. Matthies, "Obstacle detection and terrain classification for autonomous off-road navigation", *Autonomous Robots*, vol. 18, pp. 81–102, 2005.
17. C.I. Connolly, J.B. Burns, R. Weiss, "Path planning using Laplace's equation", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2102–2106, 1990.
18. J.-P. Laumond, P.E. Jacobs, M. Taix, R.M. Murray, "A motion planner for nonholonomic mobile robots", *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 577–593, 1994.
19. H. Choset, "Coverage for robotics – A survey of recent results", *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113–126, 2001.
20. A.A. Maciejewski, J.J. Fox, "Path Planning and the Topology of Configuration Space", *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 444–456, 1993.
21. K. W. Lin, Changli Mu, K. Takase, "Path Planning with Topological Map built with ID Tag and WEB Camera", *Proceedings of IEEE International Conference on Mechatronics and Automation*, pp. 1739–1744, 2006.
22. S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation", *Artificial Intelligence*, vol. 99, pp. 21–71, 1998.
23. M. A. Wesley, T. Lozano-Perez, "An algorithm for planning collision-free paths among polyhedral obstacles", *Communications of the ACM*, vol. 22, pp. 560–570, 1979.
24. D.-T. Lee, *Proximity and reachability in the plane*, PhD Thesis, University of Illinois at Urbana-Champaign, 1978.
25. I.-P. Park, J. R. Kender, "Topological direction-giving and visual navigation in large environments", *Artificial Intelligence*, vol. 78, pp. 355–395, 1995.
26. S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlingshaus, D. Hennig, T. Hofmann, M. Krell, T. Schmidt, "Map learning and high-speed navigation in RHINO: Case Studies of Successful Robot", *Artificial Intelligence and Mobile Robots Systems*, 1998.
27. S. S. Ge, *Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications (Automation and Control Engineering)*, CRC Press Taylor & Francis Group, 2006.
28. F. Fahimi, *Autonomous Robots: Modeling, Path Planning, and Control*, Springer, 2008.
29. N. H. Sleumer, N. Tschichold-Gürman, *Exact Cell Decomposition of Arrangements used for Path Planning in Robotics*, 1999.
30. D. E. Koditschek, "Exact robot navigation by means of potential functions: Some topological considerations", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 1–6, 1987.
31. L. E. Kavraki, P. Svestka, J.-C. Latombe, M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces", *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, 1996.
32. D. Zhu, J.-C. Latombe, "Constraint reformulation in a hierarchical path planner", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1918–1923, 1990.

33. J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
34. S. S. Ge, Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method", *Autonomous Robots*, vol. 13, pp. 207–222, 2002.
35. Y. K. Hwang, N. Ahuja, "A potential field approach to path planning", *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 23–32, 1992.
36. E. Rimon, D. E. Koditschek, "Exact robot navigation using artificial potential functions", *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 501–518, 1992.
37. L. Chen, H. Jin, J. Bai, H. Mao, "Path planning for mobile robots in 3D dynamic environments", *Advanced Materials Research*, vol. 403–408, ed. Li Yuan, pp. 1401–1404, 2012.
38. Q. B. Yuan, H. J. Wang, P. H. Zhu, H. Zhao, "A hybrid algorithm to solve the problem of the robot path planning", *Advanced Materials Research*, vol. 383–390, (Ed.) Wu Fan, pp. 385–389, 2012.
39. R. A. Brooks, T. Lozano-Pérez, "A Subdivision Algorithm Configuration Space for Findpath With Rotation", *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-15, 224–233, 1983.
40. A. Elfes, "Using occupancy grids for mobile robot perception and navigation", *Computer*, vol. 22, pp. 46–57, 1989.
41. P. Moscato, C. Cotta, "A gentle introduction to memetic algorithms", in: *Handbook of Metaheuristics*, pp. 105–144, 2003.
42. V. J. Lumelsky, A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape", *Algorithmica*, vol. 2, pp. 403–430, 1987.
43. M. Sharir, A. Schorr, "On shortest paths in polyhedral spaces", *SIAM Journal of Computing*, vol. 15, pp. 193–215, 1986.
44. E. Welzl, "Constructing the visibility graph for n -line segments in $O(n^2)$ time", *Information Processing Letters*, vol. 20, pp. 167–171, 1985.
45. M. H. Overmars, E. Welzl, "New methods for computing visibility graphs", *Proceedings of the 4th Annual Symposium on Computational Geometry*, pp. 164–171, 1988.
46. S. K. Ghosh, D. M. Mount, "An output sensitive algorithm for computing visibility graphs", *SIAM Journal of Computing*, vol. 20, pp. 888–910, 1987.
47. M. Pocchiola, G. Vegter, "Topologically sweeping visibility complexes via pseudotriangulations", *Discrete and Computational Geometry*, vol. 16, pp. 419–453, 1996.
48. H. Rohnert, "Shortest paths in the plane with convex polygonal obstacles", *Information Processing Letters*, vol. 23, pp. 71–76, 1986.
49. R. E. Tarjan, C. Van Wyk, "An $O(n \log \log n)$ -time algorithm for triangulating a simple polygon", *SIAM Journal of Computing*, vol. 17, pp. 143–178, 1988.
50. E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs", *Numerical Mathematics*, vol. 1, pp. 269–271, 1959.
51. M. L. Fredman, R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms", *Journal of the ACM*, vol. 34, pp. 596–615, 1987.

52. H. Edelsbrunner, *Algorithms in combinatorial geometry*, Springer-Verlag New York, Inc., 1987.
53. F. P. Preparata, M. I. Shamos, *Computational geometry: Introduction*, Springer-Verlag New York, Inc., pp. 1–35, 1985.
54. D. T. Lee, F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers", *Networks*, vol. 14, pp. 393–410, 1984.
55. J. H. Reif, J. A. Storer, *Shortest Paths in Euclidean Space with Polyhedral Obstacles*, Harvard Univ. Cambridge MA Aiken Computation Lab, 1985.
56. P. Rezende, D. Lee, Y. Wu, "Rectilinear shortest paths in the presence of rectangular barriers", *Discrete and Computational Geometry*, vol. 4, pp. 41–53, 1989.
57. R. C. Arkin, *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments*, University of Massachusetts, Amherst, 1987.
58. Y. Wang, D. Mulvaney, I. Sillitoe, E. Swere, "Robot Navigation by Waypoints", *Journal of Intelligent and Robotic Systems*, vol. 52, pp. 175–207, 2008.
59. L. P. Chew, "Planning the shortest path for a disc in $O(n^2 \log n)$ time", *Proceedings of the 1st Annual Symposium on Computational Geometry*, pp. 214–220, 1985.
60. B. Baker, "Shortest paths with unit clearance among polygonal obstacles", *Proceedings SIAM Conference on Geometric Modeling and Robotics*, pp. 207–209, 1985.
61. R. Klein, E. Langetepe, Z. Nilforoushan, "Abstract Voronoi diagrams revisited", *Computational Geometry*, vol. 42, pp. 885–902, 2009.
62. [www.cs.cornell.edu/home/chew/Delaunay.html] – P. Chew, *Voronoi Diagram / Delaunay Triangulation*, 2007.
63. M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2008.
64. M. I. Shamos, *Computational Geometry*, PhD Thesis, Yale University, New Haven, 1978.
65. P. J. Green, R. R. Sibson, "Computing Dirichlet tessellations in the plane", *Computer Journal*, vol. 21, pp. 168–173, 1978.
66. T. Ohya, M. Iri, K. Murota, "Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms", *Journal of the Operations Research Society of Japan*, vol. 27, pp. 306–337, 1984.
67. K. Sugihara, M. Iri, "Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetic", *Proceedings of the IEEE* vol. 80, pp. 1471–1484, 1992.
68. M. I. Shamos, D. Hoey, "Closest-point problems", *Proceedings of 16th Annual Symposium on Foundations of Computer Science*, pp. 151–162, 1975.
69. N. M. Amato, Y. Wu, "A randomized roadmap method for path and manipulation planning", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 113–120, 1996.

70. J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, P. Raghavan, "A Random Sampling Scheme for Path Planning", *International Journal of Robotic Research*, 1996.
71. L. E. Kavraki, *Random networks in configuration space for fast path planning*, PhD Thesis, Stanford University, 1995.
72. J.-C. Latombe, L. Kavraki, "Randomized preprocessing of configuration space for fast path planning", *Proceedings IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2138–2145, 1994.
73. R. Bohlin, L. E. Kavraki, "Path planning using Lazy PRM", *Proceedings IEEE International Conference on Robotics and Automation*, vol. 1, pp. 521–528, 2000.
74. S. M. LaValle, M. S. Branicky, K. Olson, L. Yang, "Quasi-Randomized Path Planning", *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1481–1487, 2001.
75. B. Chazelle, "The Discrepancy Method", *Proceedings of the 9th International Symposium on Algorithms and Computation*, 1998.
76. R. Geraerts, M. Overmars, "A Comparative Study of Probabilistic Roadmap Planners" in *Algorithmic Foundations of Robotics V*. vol. 7, ed.: Springer Berlin / Heidelberg, pp. 43–58, 2004.
77. V. Boor, M. H. Overmars, A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1018–1023, 1999.
78. N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces", *Proceedings of the 3rd International Workshop on the algorithmic foundations of robotics*, Houston, Texas, United States, pp. 155–168, 1998.
79. C. S. Nissoux, T. Simeon, J.-P. Laumond, "Visibility based probabilistic roadmaps", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 1316–1321, 1999.
80. S. M. LaValle, *Rapidly-exploring random trees: A new tool for path planning*, 1998.
81. E. Frazzoli, M. A. Dahleh, E. Feron, "Real-time motion planning for agile autonomous vehicles", *Journal of Guidance, Control, and Dynamics*, vol. 25, pp. 116–129, 2002.
82. S. M. LaValle, J. J. Kuffner Jr, "Randomized kinodynamic planning", *International Journal of Robotics Research*, vol. 20, pp. 378–400, 2001.
83. F. Liu, S. Liang, X. Xian, H. Bi, "Optimal path planning for mobile robot in consideration of road attributes", *ICIC Express Letters*, vol. 6, pp. 281–287, 2012.
84. J. J. Kuffner, S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, pp. 995–1001, 2000.
85. A. W. Moore, *Efficient Memory-based Learning for Robot Control*, Technical Report 209, University of Cambridge, Cambridge 1990.

86. A. Raith, M. Ehr Gott, "A comparison of solution strategies for biobjective shortest path problems", *Computers and Operations Research*, vol. 36, pp. 1299–1331, 2009.
87. S. M. LaValle, J. J. Kuffner, *Rapidly-Exploring Random Trees: Progress and Prospects*, pp. 293–308, 2000.
88. M. Strandberg, *Robot Path Planning: An Object-Oriented Approach*, PhD Thesis, 2004.
89. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.
90. J. Barraquand, B. Langlois, J.-C. Latombe, "Numerical potential field techniques for robot path planning", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 224–241, 1992.
91. M. Park, M. Lee, "A new technique to escape local minimum in artificial potential field based path planning", *Journal of Mechanical Science and Technology*, vol. 17, pp. 1876–1885, 2003.
92. M. H. Mabrouk, C. R. McInnes, "Solving the potential field local minimum problem using internal agent states", *Robots and Autonomous Systems*, vol. 56, pp. 1050–1060, 2008.
93. X. Yun, K.-C. Tan, "A wall-following method for escaping local minima in potential field based motion planning", *Proceedings of the 8th International Conference on Advanced Robotics*, pp. 421–426, 1997.
94. D. E. Koditschek, "Robot planning and control via potential functions", *The robotics review 1*, MIT Press, pp. 349–367, 1989.
95. L. Xie, H. Chen, G. Xie, "Artificial Potential Field Based Path Planning for Mobile Robots Using Virtual Water-Flow Method", in: *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*. vol. 2, D.-S. Huang, L. Heutte, M. Loog, Eds., Springer Berlin Heidelberg, pp. 588–595, 2007.
96. D. C. Conner, A. Rizzi, H. Choset, "Composition of local potential functions for global robot control and navigation", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3546–3551, 2003.
97. B. Siciliano, O. Khatib (Eds.), *Springer Handbook of Robotics*, Springer, 2008.
98. Guang Song, N. N. Amato, "Randomized Motion Planning for Car-like Robots with C-PRM", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 37–42, 2001.
99. A. Nooraliei, R. Iraj, "Robot path planning using wavefront approach with wall-following", *2nd IEEE International Conference on Computer Science and Information Technology*, pp. 417–420, 2009.
100. A. Pal, R. Tiwari, A. Shukla, "A Focused Wave Front Algorithm for Mobile Robot Path Planning", *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems*, vol. 1, pp. 190–197, 2011.
101. J. Lengyel, M. Reichert, B. R. Donald, D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware", *ACM SIGGRAPH Computer Graphics*, vol. 24, pp. 327–335, 1990.

102. O. Brock, O. Khatib, "High-Speed Navigation Using the Global Dynamic Window Approach", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 341–346, 1999.
103. R. R. Murphy, K. Hughes, E. Noll, "An Explicit Path Planner to Facilitate Reactive Control and Terrain Preferences", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2067–2072, 1996.
104. J. S. B. Mitchell, C. H. Papadimitriou, "The weighted region problem: finding shortest paths through a weighted planar subdivision", *Journal of the ACM*, vol. 38, pp. 18–73, 1991.
105. S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, 2009.
106. M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion", *Control and Cybernetics*, vol. 35, pp. 599–620, 2006.
107. R. R. Murphy, K. Hughes, A. Marzilli, E. Noll, "Integrating explicit path planning with reactive control of mobile robots using Trulla", *Robotics and Autonomous Systems*, vol. 27, pp. 225–245, 1999.
108. K. Hughes, A. Tokuta, N. Ranganathan, "trulla: An Algorithm For Path Planning Among Weighted Regions By Localized Propagations", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 469–476, 1992.
109. P. Hart, N. Nilsson, B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.
110. J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Boston 1984.
111. R. E. Korf, "Real-time heuristic search", *Artificial Intelligence*, vol. 42, pp. 189–211, 1990.
112. N. J. Nilsson, *Shakey the robot*, Technical Note 323, Artificial Intelligence Center, SRI International, Menlo Park 1984.
113. A. Stentz, "Optimal and Efficient Path Planning for Partially Known Environments", in: *Intelligent Unmanned Ground Vehicles*. vol. 388, M. H. Hebert, C. Thorpe, A. Stentz (Eds.), Springer US, pp. 203–220, 1997.
114. A. Stentz, "The focussed D* algorithm for real-time replanning", *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 2, pp. 1652–1659, 1995.
115. S. Koenig, M. Likhachev, "Fast replanning for navigation in unknown terrain", *IEEE Transactions on Robotics*, vol. 21, pp. 354–363, 2005.
116. R. E. Korf, "Depth-first iterative-deepening: an optimal admissible tree search", *Artificial Intelligence*, vol. 27, pp. 97–109, 1985.
117. X. Sun, S. Koenig, "The fringe-saving A* search algorithm: a feasibility study", *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 2391–2397, 2007.
118. S. Koenig, M. Likhachev, D. Furcy, "Lifelong planning A*", *Artificial Intelligence*, vol. 155, pp. 93–146, 2004.

119. N. Deo, C.-Y. Pang, "Shortest-path algorithms: Taxonomy and annotation", *Networks*, vol. 14, pp. 275–323, 1984.
120. S. Koenig, M. Likhachev, Y. Liu, D. Furcy, "Incremental heuristic search in AI", *AI Magazine*, vol. 25, pp. 99–112, 2004.
121. A. Stentz, "Optimal and efficient path planning for partially-known environments", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3310–3317, 1994.
122. G. Ramalingam, T. Reps, "An incremental algorithm for a generalization of the shortest-path problem", *Journal of Algorithms*, vol. 21, pp. 267–305, 1996.
123. A. Nash, K. Daniel, S. Koenig, A. Felner, "Theta*: any-angle path planning on grids", *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, 2010.
124. S. Russell, "Efficient memory-bounded search methods", *Proceedings of the 10th European Conference on Artificial Intelligence*, pp. 1–5, 1992.
125. H. H. Viet, P. H. Kyaw, T. Chung, "Simulation-based evaluations of reinforcement learning algorithms for autonomous mobile robot path planning", *Lecture Notes in Electrical Engineering*, vol. 107, pp. 467–476, 2012.
126. F. Hegel, C. Muhl, B. Wrede, M. Hielscher-Fastabend, G. Sagerer, "Understanding Social Robots", *2nd International Conferences on Advances in Computer-Human Interactions*, pp. 169–174, 2009.
127. S. Koenig, M. Likhachev, "Improved Fast Replanning for Robot Navigation in Unknown Terrain", *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 968–975, 2002.
128. D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*: Addison-Wesley Professional, 1989.
129. Z. Michalewicz, D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, 2004.
130. P. Feng, "Dynamic path planning algorithm based on chaos genetic vehicle navigation", *Applied Mechanics and Materials*, vol. 142, pp. 12–15, 2012.
131. T. Shibata, T. Fukuda, "Intelligent motion planning by genetic algorithm with fuzzy critic", *Proceedings of the IEEE International Symposium on Intelligent Control*, pp. 565–570, 1993.
132. X. Liao, G. G. Wang, "Evolutionary path planning for robot assisted part handling in sheet metal bending", *Robotics and Computer-Integrated Manufacturing*, vol. 19, pp. 425–430, 2003.
133. M. Zhao, N. Ansari, E. S. H. Hou, "Mobile manipulator path planning by a genetic algorithm", *Journal of Robotic Systems*, vol. 11, pp. 143–153, 1994.
134. P. Bessiere, J.-M. Ahuactzin, E.-G. Talbi, E. Mazer, "The Ariadne's clew algorithm: Global planning with local methods", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 1373–1380, 1993.
135. J. Xiao, Z. Michalewicz, L. Zhang, K. Trojanowski, "Adaptive Evolutionary Planner/Navigator for Mobile Robots", *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 19–28, 1997.

136. Z. Michalewicz, J. Xiao, "Evaluation of paths in evolutionary planner/navigator", *Proceedings of the International Workshop on Biologically Inspired Evolutionary Systems*, pp. 45–52, 1995.
137. J. Xiao, "The Evolutionary Planner/Navigator in a mobile robot environment" in: *Handbook of Evolutionary Computation*, D. B. Fogel, T. Back, Z. Michalewicz (Eds.), Taylor & Francis, 1997.
138. J. Xiao, Z. Michalewicz, L. Zhang, "Evolutionary Planner/Navigator: Operator Performance and Self-Tuning", *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 366–371, 1996.
139. I. Koryakovskiy, N. X. Hoai, K. M. Lee, "A genetic algorithm with local map for path planning in dynamic environments", *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1859–1860, 2009.
140. K. Sugihara, J. Smith, "Genetic Algorithms for Adaptive Motion Planning of an Autonomous Mobile Robot", *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 138–143, 1997.
141. Y. Liang, L. Xu, "Global path planning for mobile robot based genetic algorithm and modified simulated annealing algorithm", *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 303–308, 2009.
142. O. Castillo, L. Trujillo, P. Melin, "Multiple Objective Genetic Algorithms for Path-planning Optimization in Autonomous Mobile Robots", *Soft Computing*, vol. 11, pp. 269–279, 2007.
143. H. Esmaeilzadeh, N. Shahidi, M. Abdollahi, C. Lucas, "Memetic Algorithm Based Path Planning for a Mobile Robot", *International Journal of Information Technology*, vol. 1, pp. 100–103, 2005.
144. L. Shidong, D. Mingyue, C. Chao, J. Lei, "Efficient Path Planning Method Based on Genetic Algorithm Combining Path Network", *Proceedings of 4th International Conference on Genetic and Evolutionary Computing*, pp. 194–197, 2010.
145. Y. Soh Chin, V. Ganapathy, C. Lim Ooi, "Improved genetic algorithms based optimum path planning for mobile robot", *11th International Conference on Control Automation Robotics & Vision*, pp. 1565–1570, 2010.
146. T. Davies, A. Jnifene, "Multiple Waypoint Path Planning for a Mobile Robot using Genetic Algorithms", *Proceedings of IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, pp. 21–26, 2006.
147. S. Yang, S. N. Jat, "Genetic Algorithms With Guided and Local Search Strategies for University Course Timetabling", *IEEE Transactions on Systems, Man, and Cybernetics*, Part C: *Applications and Reviews*, vol. 41, pp. 93–106, 2011.
148. M. Gao, J. Xu, J. Tian, "Mobile Robot Global Path Planning Based on Improved Augment Ant Colony Algorithm", *Second International Conference on Genetic and Evolutionary Computing*, pp. 273–276, 2008.
149. C.-K. Loo, M. Rajeswari, E. K. Wong, M. V. C. Rao, "Mobile Robot Path Planning Using Hybrid Genetic Algorithm and Traversability Vectors

- Method", *Intelligent Automation and Soft Computing*, vol. 10, pp. 51–63, 2004.
150. L. Xu, Y. Liang, "Global path planning for mobile robot based genetic algorithm and modified simulated annealing algorithm", *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 303–308, 2009.
 151. G. Mayley, "Landscapes, learning costs, and genetic assimilation", *Evolutionary Computation*, vol. 4, pp. 213–234, 1996.
 152. W. E. Hart, R. K. Belew, "Optimizing an Arbitrary Function is Hard for the Genetic Algorithm", *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 190–195, 1991.
 153. U. Nehmzow, "Physically Embedded Genetic Algorithm Learning in Multi-Robot Scenarios: The PEGA algorithm", *Proceedings of Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 2002.
 154. H.-C. Huang, C.-C. Tsai, S.-C. Lin, "SoPC-based parallel elite genetic algorithm for global path planning of an autonomous omnidirectional mobile robot", *Proceedings of IEEE international conference on Systems, Man and Cybernetics*, pp. 1959–1964, 2009.
 155. A. Ghorbani, S. Shiry, A. Nodehi, "Using Genetic Algorithm for a Mobile Robot Path Planning", *International Conference on Future Computer and Communication*, pp. 164–166, 2009.
 156. N. Sadati, J. Taheri, "Genetic algorithm in robot path planning problem in crisp and fuzzified environments", *IEEE International Conference on Industrial Technology*, vol. 1, pp. 175–180, 2002.
 157. C. Huahua, X. Zezhong, "Path Planning Based on A New Genetic Algorithm", *International Conference on Neural Networks and Brain*, pp. 788–792, 2005.
 158. N. Sadati, J. Taheri, "Solving robot motion planning problem using Hopfield neural network in a fuzzified environment", *Proceedings of the IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 1144–1149, 2002.
 159. Z. Yong, Z. Lin, Z. Xiaohua, "Mobile Robot Path Planning Base on the Hybrid Genetic Algorithm in Unknown Environment", *8th International Conference on Intelligent Systems Design and Applications*, vol. 2, pp. 661–665, 2008.
 160. L. Chuan-ling, L. De-qi, Y. Jing-Yu, "A Novel Method of Mobile Robot Path Planning Based on Quantum Genetic Algorithm", *Chinese Conference on Pattern Recognition*, pp. 1–5, 2010.
 161. W. Rui, G. Ning, X. Fenghong, M. Jianlin, "An improved quantum genetic algorithm with mutation and its application to 0–1 knapsack problem", *International Conference on Measurement, Information and Control*, vol. 1, pp. 484–488, 2012.
 162. L. Chuan-ling, W. Ming-hua, Y. Jing-ju, "An improved quantum genetic algorithm and its application in path planning of mobile robots", *International Conference on Computer Application and System Modeling*, vol. 7, pp. V7-413–V7-417, 2010.

163. K. F. Man, K. S. Tang, S. Kwong, *Genetic Algorithms: Concepts and Designs*, Springer-Verlag New York, Inc., 1999.
164. W. Chunmiao, Y. C. Soh, W. Han, W. Hui, "A hierarchical genetic algorithm for path planning in a static environment with obstacles", *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 500–505, 2002.
165. B. Friesleben, P. Merz, "A genetic local search algorithm for solving the symmetric and asymmetric traveling salesman problem citation", *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 616–621, 1996.
166. J. Baohua, Z. Liang, "A Path Planning Method of Contingency Logistics Based on Max-Min Ant System", *Asia-Pacific Conference on Wearable Computing Systems*, pp. 311–314, 2010.
167. M. Fatih Tasgetiren, P. N. Suganthan, P. Quan-ke, L. Yun-Chia, "A genetic algorithm for the generalized traveling salesman problem", *IEEE Congress on Evolutionary Computation*, pp. 2382–2389, 2007.
168. Z. Yao, L. Ma, "A Static Environment-Based Path Planning Method by Using Genetic Algorithm", *International Conference on Computing, Control and Industrial Engineering*, vol. 2, pp. 405–407, 2010.
169. D. H. Wolpert, W. G. Macready, "No free lunch theorems for optimization", *IEEE Transactions On Evolutionary Computation*, vol. 1, pp. 67–82, 1997.
170. B. Wilcox, R. Ambrose, B. Reed, L. Matthies, D. Lavery, D. Korsmeyer, *Robotics, Tele-Robotics and Autonomous Systems. Technology Area 04*, National Aeronautics and Space Administration, 2012.
171. M. Koppen, D. H. Wolpert, W. G. Macready, "Remarks on a recent paper on the "no free lunch" theorems", *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 295–296, 2001.
172. H. Yu-Chi, D. L. Pepyne, "Simple explanation of the no free lunch theorem of optimization", *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 5, pp. 4409–4414, 2001.
173. T. Bäck, "Order Statistics for Convergence Velocity Analysis of Simplified Evolutionary Algorithms", *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, pp. 91–102, 1994.
174. T.-P. Hong, H.-S. Wang, W.-C. Chen, "Simultaneously Applying Multiple Mutation Operators in Genetic Algorithms", *Journal of Heuristics*, vol. 6, pp. 439–455, 2000.
175. P. D. Surry, N. J. Radcliffe, "Inoculation to initialise evolutionary search Evolutionary Computing", *Lecture Notes in Computer Science*, vol. 1143, pp. 269–285, 1996.
176. W. E. Hart, *Adaptive global optimization with local search*, University of California at San Diego, 1994.
177. N. Krasnogor, *Studies on the Theory and Design Space of Memetic Algorithms*, PhD Thesis, University of the West of England, 2002.
178. M. W. S. Land, *Evolutionary algorithms with local search for combinatorial optimization*, PhD Thesis, University of California, San Diego, 1998.

179. I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Frommann-Holzboog, 1973.
180. D. Goldberg, R. Lingle, "Alleles, loci, and the traveling salesman problem", *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp. 154–159, 1985.
181. R. Kicinger, T. Arciszewski, K. Jong, "Evolutionary computation and structural design: A survey of the state-of-the-art", *Computers & Structures*, vol. 83, pp. 1943–1978, 2005.
182. D. Fogel, *Evolutionary Computation: The Fossil Record*, Wiley-IEEE Press, 1998.
183. C. R. Houck, J. A. Joines, M. G. Kay, J. R. Wilson, "Empirical investigation of the benefits of partial lamarckianism", *Evolutionary Computation*, vol. 5, pp. 31–60, 1997.
184. J. M. Baldwin, "A New Factor in Evolution", *The American Naturalist*, vol. 30, pp. 441–451, 1896.
185. P. Turney, "How to shift bias: Lessons from the baldwin effect", *Evolutionary Computation*, vol. 4, pp. 271–295, 1996.
186. L. D. Whitley, V. S. Gordon, K. E. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization", *Lecture Notes in Computer Science*, vol. 866, pp. 5–15, 1994.
187. F. Gruau, D. Whitley, "Adding learning to the cellular development of neural networks: Evolution and the Baldwin effect", *Evolutionary Computation*, vol. 1, pp. 213–233, 1993.
188. L. Zhang, Y. Wang, B. Wang, Q. Deng, H. Chen, "Path planning for mobile robot based on ACA-GA", *Applied Mechanics and Materials*, vol. 135–136, pp. 673–677, 2012.
189. M. Dorigo, C. Blum, "Ant colony optimization theory: A survey", *Theoretical Computer Science*, vol. 344, pp. 243–278, 2005.
190. M. Dorigo, *Optimization, Learning and Natural Algorithms*, PhD Thesis, Politecnico di Milano, Italy, 1992.
191. D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, B. Baesens, "Classification With Ant Colony Optimization", *IEEE Transactions on Evolutionary Computation*, vol. 11, pp. 651–665, 2007.
192. B. Yagmahan, M. M. Yenisey, "Ant colony optimization for multi-objective flow shop scheduling problem", *Computers & Industrial Engineering*, vol. 54, pp. 411–420, 2008.
193. P. Toth, D. Vigo, "Models, relaxations and exact approaches for the capacitated vehicle routing problem", *Discrete Applied Mathematics*, vol. 123, pp. 487–512, 2002.
194. R. Bent, P. V. Hentenryck, "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows", *Computers & Operations Research*, vol. 33, pp. 875–893, 2006.
195. H. Ramalhinho-Lourenço, D. Serra, *Adaptive approach heuristics for the generalized assignment problem*, 1998.

196. V. Maniezzo, M. Milandri, "An Ant-Based Framework for Very Strongly Constrained Problems", *Lecture Notes in Computer Science*, vol. 2463, pp. 115–148, 2002.
197. Z.-q. Wen, Z.-x. Cai, "Global path planning approach based on ant colony optimization algorithm", *Journal of Central South University of Technology*, vol. 13, pp. 707–712, 2006.
198. G.-Z. Tan, H. He, A. Sloman, "Ant Colony System Algorithm for Real-Time Globally Optimal Path Planning of Mobile Robots", *Acta Automatica Sinica*, vol. 33, pp. 279–285, 2007.
199. T. White, S. Kaegi, T. Oda, "Revisiting Elitism in Ant Colony Optimization", *Lecture Notes in Computer Science*, vol. 2723, pp. 122–133, 2003.
200. T. Stützle, H. H. Hoos, "MAX-MIN Ant System", *Future Generation Computer Systems*, vol. 16, pp. 889–914, 2000.
201. B. Jin, L. Zhang, "A Path Planning Method of Contingency Logistics Based on Max-Min Ant System", *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems*, pp. 311–314, 2010.
202. W. J. Gutjahr, "A generalized convergence result for the graph-based ant system metaheuristic", *Probability in the Engineering and Informational Science*, vol. 17, pp. 545–569, 2003.
203. X.-M. Hu, J. Zhang, Y. Li, "Orthogonal Methods Based Ant Colony Search for Solving Continuous Optimization Problems", *Journal of Computer Science and Technology*, vol. 23, pp. 2–18, 2008.
204. H. Qingyao, W. Qiang, Z. Xiaoguang, X. Jin, "Path planning of mobile robot based on improved ant colony algorithm", *International Conference on Consumer Electronics, Communications and Networks*, pp. 531–533, 2011.
205. L. Joon-Woo, K. Jeong-Jung, C. Byoung-Suk, L. Ju-Jang, "Improved Ant Colony Optimization algorithm by potential field concept for optimal path planning", *8th IEEE-RAS International Conference on Humanoid Robots*, pp. 662–667, 2008.
206. L. Joon-Woo, K. Jeong-Jung, L. Ju-Jang, "Improved Ant Colony Optimization algorithm by path crossover for optimal path planning", *IEEE International Symposium on Industrial Electronics*, pp. 1996–2000, 2009.
207. F. Xiaoping, L. Xiong, Y. Sheng, Y. Shengyue, Z. Heng, "Optimal path planning for mobile robots based on intensified ant colony optimization algorithm", *Proceedings of IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, vol. 1, pp. 131–136, 2003.
208. L. Joon-Woo, L. Dong-Hyun, L. Ju-Jang, "Global path planning using improved ant colony optimization algorithm through bilateral cooperative exploration", *Proceedings of the 5th IEEE International Conference on Digital Ecosystems and Technologies Conference*, pp. 109–113, 2011.
209. W. Yongfen, S. Lei, L. Qing, C. Yao, "Novel path planning of robots based on bidirectional ant colony algorithm", *6th International Conference on Natural Computation*, vol. 5, pp. 2403–2406, 2010.
210. V. Nguyen Hoang, V. Ngo Anh, L. SeungGwan, C. TaeChoong, "Obstacle Avoidance Path Planning for Mobile Robot Based on Multi Colony Ant

- Algorithm", *1st International Conference on Advances in Computer-Human Interaction*, pp. 285–289, 2008.
211. M. Dorigo, L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem", *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, 1997.
 212. M. Middendorf, F. Reischle, H. Schmeck, "Information Exchange in Multi Colony Ant Algorithms", *Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pp. 645–652, 2000.
 213. B. Artyushenko, "Analysis of global exploration of island model genetic algorithm", *10th International Conference - The Experience of Designing and Application of CAD Systems in Microelectronics*, pp. 280–281, 2009.
 214. D.-H. Nguyen, J. Jyh Ching, "A refined ant colony algorithm for optimal path planning", *International Conference on System Science and Engineering*, pp. 125–130, 2011.
 215. B. A. Garro, H. Sossa, R. A. Vazquez, "Evolving ant colony system for optimizing path planning in mobile robots", *Electronics, Robotics and Automotive Mechanics Conference*, pp. 444–449, 2007.
 216. B. A. Garro, H. Sossa, R. A. Vazquez, "Path Planning Optimization Using Bio-Inspired Algorithms", *5th Mexican International Conference on Artificial Intelligence*, pp. 319–330, 2006.
 217. Y. Along, G. Lixin, L. Yanping, "An Improved Ant Colony System Algorithm for Optimal Path Planning Problem of Mobile Robots", *Second International Conference on Computer Modeling and Simulation*, vol. 1, pp. 526–530, 2010.
 218. J. Kennedy, R. Eberhart, "Particle swarm optimization", *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.
 219. X. Wu, Y. Ming, J. Wang, "An improved path planning approach based on Particle Swarm Optimization", *11th International Conference on Hybrid Intelligent Systems*, pp. 157–161, 2011.
 220. M. Saska, M. Macas, L. Preucil, L. Lhotska, "Robot Path Planning using Particle Swarm Optimization of Ferguson Splines", *IEEE Conference on Emerging Technologies and Factory Automation*, pp. 833–839, 2006.
 221. J. Ye, R. Qu, "Fairing of parametric cubic splines", *Mathematical and Computer Modelling*, vol. 30, pp. 121–131, 1999.
 222. L. Li, G. Dunwei, "Robot Path Planning in Unknown Environments Using Particle Swarm Optimization", *Fourth International Conference on Natural Computation*, vol. 4, pp. 422–426, 2008.
 223. K. Hwan-il, L. Byunghye, K. Kabil, "Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm", *Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, vol. 2, pp. 1002–1004, 2008.
 224. M.K. Habib, H. Asama, "Efficient method to generate collision free paths for an autonomous mobile robot based on new free space structuring approach", *Proceedings IROS '91. IEEE/RSJ International Workshop on Intelligent Robots and Systems*, vol. 2, pp. 563–567, 1991.

- 225. L. Qing, T. Yong, W. Lijun, Z. Chao, Y. Yixin, "A Specialized Particle Swarm Optimization for global path planning of mobile robots", *Third International Workshop on Advanced Computational Intelligence*, pp. 271–276, 2010.
- 226. Q. Zhang, G. Gu, "Path Planning Based on Improved Binary Particle Swarm Optimization Algorithm", *IEEE Conference on Robotics, Automation and Mechatronics*, pp. 462–466, 2008.
- 227. M. Qianzhi, L. Xiujuan, Z. Qun, "Mobile Robot Path Planning with Complex Constraints Based on the Second-Order Oscillating Particle Swarm Optimization Algorithm", *World Congress on Computer Science and Information Engineering*, vol. 5, pp. 244–248, 2009.
- 228. B. Liu, L. Wang, Y.-H. Jin, F. Tang, D.-X. Huang, "Improved particle swarm optimization combined with chaos", *Chaos, Solitons & Fractals*, vol. 25, pp. 1261–1271, 2005.
- 229. K. Lei, Y. Qiu, Y. He, "A novel path planning for mobile robots using modified particle swarm optimizer", *1st International Symposium on Systems and Control in Aerospace and Astronautics*, pp. 981–984, 2006.
- 230. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- 231. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.
- 232. C. Andrieu, N. de Freitas, A. Doucet, M. Jordan, "An Introduction to MCMC for Machine Learning", *Machine Learning*, vol. 50, pp. 5–43, 2003.
- 233. M. Hui, T. Yu-Chu, "Robot path planning in dynamic environments using a simulated annealing based approach", *10th International Conference on Control, Automation, Robotics and Vision*, pp. 1253–1258, 2008.
- 234. G. Meijuan, T. Jingwen, "Path Planning for Mobile Robot Based on Improved Simulated Annealing Artificial Neural Network", *Third International Conference on Natural Computation*, vol. 3, pp. 8–12, 2007.
- 235. M.J.D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives", *The Computer Journal*, vol. 7, pp. 155–162, 1964.
- 236. F. Janabi-Sharifi, D. Vinke, "Integration of the artificial potential field approach with simulated annealing for robot path planning", *Proceedings of IEEE International Symposium on Intelligent Control*, pp. 536–541, 1993.
- 237. Z. Qidan, Y. Yongjie, X. Zhuoyi, "Robot Path Planning Based on Artificial Potential Field Approach with Simulated Annealing", *Sixth International Conference on Intelligent Systems Design and Applications*, vol. 2, pp. 622–627, 2006.
- 238. H. Martinez-Alfaro, D.R. Flugrad, "Collision-free path planning for mobile robots and/or AGVs using simulated annealing", *IEEE International Conference on Systems, Man, and Cybernetics, Humans, Information and Technology*, vol. 1, pp. 270–275, 1994.
- 239. Z. Qi, M. Jiachen, L. Qiang, "Path planning based quadtree representation for mobile robot using hybrid-simulated annealing and ant colony optimization

- algorithm", *10th World Congress on Intelligent Control and Automation*, pp. 2537–2542, 2012.
240. Y. Hong, W. Zhong-Min, "Path planning of mobile robot based on compound shape and simulated annealing hybrid algorithm", *IEEE International Conference on Robotics and Biomimetics*, pp. 186–189, 2005.
241. M. Ghallab, D. Nau, P. Traverso, *Automated Planning. Theory and Practice*, Morgan Kaufmann Publishers, 2004.