Contents

1 Introduction

Automatization of complex tasks require robots to be able to plan its movements not only by what is the shortest route but potentially considering a large number of different trade-offs. If we want a robot to be useful outside of the lab and factory floor it would have to be able to operate in environments that can change unpredictably or are to a large degree unknown, and it would have to be able to adapt with presicion and speed to unforeseen changes in situation and environment.

In most real-world tasks there will be multiple, often conflicting measures of how well the task has been performed. When these measures conflict there will be trade-offs between different solutions. How to best solve problems where such conflicts exist, and what considerations must be made when trying to solve such tasks is the field of multi-objective optimization.

One of the most common and basic task in robot automatization is that of path planning. In its strictest form path-planning is solving the following problem: Given an initial situation, we'd want the robot to plan the motions needed to get to a different pose or position in the best way possible. But adapting the plan to changes in the environment is also considered a part of path-planning. Furthermore, there might be other path or movement planning related objectives than than simply moving to a different point in space - cover a certain area, for example.

Evolutionary algorithms, algorithms that optimize through a population of evolving solutions, especially the subfield of genetic algorithms, have acheieved some popularity in both multi-objective optimization and in path-planning. While multi-objective optimization exploit evolutionary algorithms' inherent property of not only giving one, but rather a large number of near-optimized solutions[?], path-planning problems make good use of evolutionary algorithms' ability to explore vast search spaces while not getting stuck in local maxima[?] and also their ability to continue to improve and adapt solutions over time in upredictable environments[?].

This essay attempts to give an overview over the fields of path-planning, evolutionary algorithms and multi-objective optimization, including pointers to recent work in these fields, especially where they intersect. In tradition with other literature relating to evolutionary algorithms, algorithms which are not population-based or otherwise based on principles similar to evolutionary algorithms are called 'classical' to separate them from evolutionary variants.

2 Classical Path-planning

In general the problem of path-planning is to find a continuous path for some moving object though an environment such that it achieves certain objectives. Usually the objective is simply to get to some other point in the environment while travelling the shortest distance possible. As the object in question is usually a robot, it will from here on be referred to as the robot, even though these algorithms have plenty of applications outside of robotic, see the introductory part of [16].

2.1 Configuration space

When considering the movements of a robot it is useful to do so in terms of the mechanics and degrees of freedom of the robot. The robots base position, its rotation and the rotation or translation of all its links if any is called the configuration of the robot. The set of all possible robot configurations constitute the robots configuration space, or C-space [19]. Working in C-space simplifies the path-planning problem, because the robot here consists of a single point. All obstacles in the robots environment then becomes obstacles in C-space, regions of C-space that are infeasible robot configurations. If the robot is a simple translational robot, i.e. that it does not rotate, have no links and can move in the same speed in any direction at any time, then C-space is identical to world space except obstacles in C-space are "padded" with the shape of the robot. For robots with many complex parts making an precise translation into C-space can become time-consuming, but it is often relatively easy to at least find a worst-case estimate, which is sufficient for most purposes.

A curve through C-space would constitute a transition between configurations. A configuration is feasible if it is in unobstructed C-space, that is, it is not inside any obstacles, but a configuration transition might not be feasible even if no point on the curve is inside an obstacle. That is because there might be dynamic constraints limiting how the robot is able to move which are not captured in C-space, such as an acceleration constraint. However, it is common to assume that any straight line segment not intersecting any obstacle is a feasible configuration transition for path-planning purposes.

2.2 Overview of Algorithms

This section will give a brief overview of the main families of algorithms that have been developed to solve path-planning problems. A more detailed introduction overview of the algorithms can be found in [15] and [16]. The most basic distinction between the different algorithms is between local and global planners. While global planners find a complete path solving the task based on information on the entire environment, local planners only proposes a way to get onward from the current state given the immediately surrounding environment. By applying a local planning algorithm repeatedly one may then plan a full path. Only one local planner - the artificial potential fields algorithm - will be discussed here.

2.2.1 Global Planners

Global planners find solutions to path-planning problems by creating a suited representation of C-space and then searching for solutions in that representation. Usually the representation is some kind of graph with feasible configurations as nodes and feasible configuration transitions as edges between these nodes. Finding the shortest path from start to goal is then a matter of running a graph searching algorithm such as A* from the start node to the goal node. This graph is usually called a roadmap.

The different algorithms then differ in how the nodes are distributed in C-space, and how they are connected. A main distinction here is whether the algorithm

can guarantee that the graph will contain a path from start to goal if it such a suloution exists, or rather, under what conditions such a guarantee can be given.

2.2.2 Combinatorial path-planners

Combinatorial path-planners are algorithms that create a roadmap by analysing the geometry of the entire C-space and all obstacles. This includes the visibility graph, voronoi diagram and silhuette algorithms.

The visibility graph has a node for each corner on each obstacle, and an edge for each pair of nodes that are mutually "visible", e.g. the straight line between them is a feasible transition [Ó'Dúnlaing et al., 1983]. This algorithm is often used used as a reference algorithm because the best solution in the visibility graph is guaranteed to be the shortest path possible in C-space.

Variants of the visibility graph, such as the art gallery algorithm described in [20] ammend the performance problems of the visibility graph by reducing the number of nodes and edges used, while still keeping the main concept of making sure that all parts of unobstructed C-space are "visible". However, some of these lose the property of containing the shortest possible path, as is the case with the art gallery algorithm.

The silhuette algorithm divides the C-space into convex slices of unobstructed C-space. Each edge between slices becomes a node in the roadmap, and each node is connected to each other node in the two slices it lies between. All of these edges will be feasible transitions, because each point in a convex set can "see" each other point in the set [Canny, 1988; 1987].

A voronoi diagram path-planning algorithm finds all points in C-space that are equidistant from the two or more nearest obstacles. This set of points is forms the voronoi diagram of the obstacles. The resulting roadmap has a node for each corner in the diagram and an edge for each edge between corners in the diagram [1, 4]. This roadmap will not find the shortest path in general, but it has the property of generating safe solutions - solutions which keep maximum distance from the nearest obstacles.

All the combinatorial algorithms have in common that the way they create the graph guarantees that you can find a feasible solution if one exists. Thus they are said to be *complete*. However, for many-dimensional C-spaces or environments with high geometrical complexity they can become impractically slow, because of the complexity of creating the roadmap, or because the graph becomes too large to search efficiently.

2.2.3 Sampling-based global Planners

Sampling-based planners do not analyze C-space geometrically in the way combinatorial planners do, but rather sample them, and cannot guarantee to find solutions in the same way the complete planners do. However, they do guarantee that if enough samples are included they can find a solution if one exists. This is termed *resolution-complete* or *probabilistically complete*, depending on wether the sampling is done in a systematic fashion or by some probabilistic method, respectively.

Though they are a notch less reliable than the combinatorial algorithms, they are less dependent upon the complexity and accuracy of the obstacles geometry, because they only need to be able to find out whether the points they sample are obstructed or not, while the combinatorial algorithms need the surface of the obstacles to be defined in a manner they can process. Furthermore, real-world geometry is usually so complex you have to simplify it in order to make combinatorial algorithms run with decent performance, and then the completeness guarantee given by those algorithms is weakened in comparison to the one given by sampling-based planners.

The simplest way to sample a space is to sample evenly along every axis, creating a grid. The grid will have obstructed and unobstructed nodes, and one would assume that all straight lines between two neighbouring unobstructed nodes are feasible transitions. This is a popular approach in many situations, because it enables efficient memory storage and makes the underlying search algorithm easy to implement. The resolution neccesary to find paths through tight passages will often give rise to an enourmous amount of nodes, though, and memory usage quickly becomes very inefficient when C-space has high dimensionality. These problems can be amended by using an adaptive resolution sampling scheme like quad-trees, effectively taking more samples near the borders of obstacles.

A different approach to sampling C-space is to take random samples. One then usually discards obstructed samples and keep sampling until a certain number of unobstructed samples are found. These samples are then connected to other samples nearby to form a roadmap. Because of that these algorithms are called probabilistic roadmaps (PRM).

Uniformly random sampling will result in many of the same trade-offs between graph size and completeness as the grid-based algorithms, and so methods for adaptive sampling and determining exactly which nodes to connect have been developed that to a large degree ammends this [3]. However, this comes ate the cost of increasing the pre-processing time, the time needed to construct the roadmap, therfore some algorithms delay the connection of nodes until needed by the search algorithm [?].

2.2.4 Artificial Potential Fields

Artificial potential fields (APF) is an local path-planning algorithm based on the idea of regarding the goal as having an attractive potetial field function and the obstacles as having a repulsive potential field function associated to them that can be evaluated all over C-space. An artificial potential field function made from summing up all the individual field functions can then be calculatedm and a path is then generated by doing a gradient decent of this function.

First formulated in [?], it was soon discovered that it was impossible to avoid the APF having local minima in addition to the goal. Since gradient decent only finds a local minimum, one cannot guarantee that this algorithm will find a way to the goal in all situations. A number of workarounds for this has been proposed since then, improving the performance of the algorithm by making local minima more unlikely and using techniques for helping the gradient decent escape local minima [?], but it is still an unreliable algorithm in many situaations. Also, the path found by an APF algorithm winn seldom be the shortest path to the

goal, since it will often have a tendency to only steer away from obstacles when getting near them, thus taking slight detours.

While unreliable and most likely not optimal, it is still widely used because it is easy to implement and because it has low complexity - for each point along the path one only needs to evaluate a simple gradient function of the goal and each obstacle nearby once. It can also be very useful as a local obstacle avoidance algorithm when travelling a path found by a global planner.

2.3 Path-planning in unknown and dynamic Environments

So far it has been assumed that the environment is entirely static and know beforehand. There exists valuable applications for planners that work in such an environment, like motion planning for industrial robots in a factory production line or an automated storage facility, but they are easily outnumbered by the potential applications of path-planners able to work in environments that are not completely known and may be changing.

Something that cannot be known before the planned path is set in motion will in this text be referred to as unknown. Thus when there are unknown elements in the environment a feasible plan may be invalidated at a later time because a new obstacle has been sensed or an obstacle has moved in an unpredictable way. So in an unknown environment there will be a need to re-plan in addition to the initial planning.

A dynamic environment, on the other hand, is an environment with one or more moving obstacles. Such an environment is still be completely known if the dynamics of each obstcle is known - that is, it is know exactly how they will move in the future. Strictly speaking a plan may not be invalidated in such an environment, it may only be wrongly assumed feasible by the planning algorithm.

Combining these two factors we have four different scenarios: a known static environment, which is what has been discussed so far, an unknown static environment, where obstacles can be sensed later but never dissapear or move. A known dynamic environment might be occur for example when several robots are collaborating or working in an environment together with moving machinery, or in space navigation. In the last and most general scenario, an unknown dynamic environment, there might be obstacles that the robot is unaware of a the time of initial planning, and any number of obstacles might be in unpredictable motion.

2.3.1 Unknown static Environments

In its most extreme, a statiuc environment might be completely unknown. In that case the task of the robot is usually to gather a map of the world while moving though it. While there are certainly path-planning aspects to that task most of the difficulties involved are more low-level, and it has become more of a separate field of research called simultaneous localization and mapping, or SLAM. Some attention has been paid navigating these environments with gols beoynd mapping in mind, thought, for example [8].

Path-planning in the case where there are only a few “surprises” in the environent is usually solved by re-running the path-planning when a obstacle appears in the

planned path. A common improvement her is to re-use the roadmap generated during the initial planning and only update where neccesary because of the new obstacle. This can save a lot of time, because in many algorithms the construction of the roadmap is the most expensive part.

2.3.2 Environments with known Dynamics

A fair amount of attention has been given to solving the path-planning problem in environments with known dynamics. The computational complexity of such problems is explored in [22]. In [2] a kind of PRM that connects nodes during the global search using a local search algorithm that considers the time used, and assigns a time of arrival to each node until it it reaches the goal.

[9] introduces the state-time space, including not only a dimension of time, but also multiple dimensions of dynamics in order to give a complete map of all possible states of the robot at all possible points in time. In state-time space dynamic constraints are represented in exactly the same as physical obstacles, it is a kind of extended C-space, and enables the use of path-planning algorithms made for static environments to be applied to known dynamic environments with only minor changes, given that they are still effective in spaces of significantly higher dimensionality. [13] and later [24] explores two different approaches to PRMs in state-time space.

2.3.3 Environments with unknown Dynamics

When the environment can change unpredicatbly it may no longer be possible to plan a path from start to goal that is guaranteed colission-free. While it has been shown in [26] that it is possible to give such a guarantee given a path and an upper bound on the velocity of any obstacle, path-planners for unknown dynamic environments usually simply assume that no obstacles will move, and then re-run the planner when an obstacle is in a position that coincides with the path planned. When doing so it is important to be able to come up with new paths quickly. One recent example of such a path planner is [18], which generates a hierarcial roadmap with deeper levels in the hierarchy having increasingly large numbers of alternate paths between nodes.

3 Evolutionary Algorithms

Evolutionary algorithms give a general framework for optimization algorithms based on the basic consepts of evolution in nature, and has adopted a lot of terminology from evolutionary biology and genetics. The general idea is to cross a set of solutions to a problem, letting information about which solutions are better and wich are worse guide the creation of iteratively better sets of solutions. In evolutionary algorithm parlance one create an initial population, let the individuals create offspring through crossover and mutation, make that new generation the new population and repeat.

Here, the population is the set of solutions, an idividual is a solution and fitness is a measure of how good a solution is. A generation is a set of solutions created based on the same base set of solutions and mutation and crossover are

the operations they are created by using one solution or a tuple of solutions, respectively. Besides good evidence that evolution has worked out pretty well for nature, there have been done some work to explain why evolutionary algorithms can optimize efficiently even when little is known of the problem to be optimized except for a fitness function and a good way to represent a solution. A summary of that explanation can be found in [14].

3.1 Overview

This section will give a short overview of evolutionary algorithms, with focus on the variants most relevant for path-planning. For a more in-depth explanation of evolutionary algorithms and its variants, see [7] and also the chapters introducing single-objective evolutionary algorithms in [6] or [5].

As outlined above, evolutionary algorithms (EAs) work on a set of solutions instead of trying to find the best solution directly. Initially, none of the solutions in the set are likely to be especially good, and in some cases even solutions known to be invalid are included. But, taken together the solutions give an indication as to what combination of parameters makes a solution better or worse. In each iteration the population is partially or completely replaced by a new generation of solutions. By relating the fitness - which ideally is a measure of distance from optimality - of each individual to its chances to propagate its parameter values to the next generation the population as a whole is gradually steered towards more optimal solutions.

If it is possible for a solution to survive unaltered into the next generation the EA is elitist - the elite individual are kept through generations. But the main way to propagate parameters to the next generations are through the operations of crossover and mutation which add new solutions to the population.

Crossover creates a mix of two solutions and so the new solution will be somewhere between or near its parents in the search space. Good solutions are selected for parenthood more often than bad solutions, so the algorithm mostly tries out new solutions that are somwhere between known good solutions. Mutation displaces a solution in a random manner in the search space.

A mutation operator is intended to introduce new variation to the population, in contrast to the crossover operator which decreases variation by blending solutions together. The mutation operator is said to explore - by searching in some random direction, while the cross-over operator, in cooperation with fitness-based selection, exploit - by searching between or near good solutions. Some EAs have no cross-over operator, and creates the next generation from mutation only. In that case the algorithm relies on which solutions are selected for mutation to do the exploitation and steer the population towards optimality.

3.1.1 Genetic Algorithms

Genetic algorithms (GAs) was the first variant of population-based optimization to appear, with early works including [12] and [10]. In its original form the solution representation was a fixed-length binary string, inspired by the strings of DNA that make up genetic material, which also inspired the crossover and mutation operators used in genetic algorithms.

In binary coded GAs crossover is usually done by taking two individuals and creating offspring where different segments of its binary string is taken from different parents. There are many variations on how to do this, but basically each bit in the resulting string is inherited from either one parent or the other. The mutation operator takes a solution and flips one or more random bits in the binary string.

This binary string representation proved to be cumbersome in many problems and so a representation of a solution as a real-valued string came in use. This representation does not work so well with the simple crossover and mutation operators of binary GAs, and so new operators was also developed that aimed at keeping the properties exploration and exploitation based on arithmetics an statistics.

3.1.2 Genetic Programming

Most evolutionary algorithms represent their solutions as some sort of fixed or variable length string of parameters, which is then used by a pre-difined formula or algorithm to solve the problem. However, in some cases there might be so little known about the problem that no such algorithm is known, or there might be several candidate algorithms.

In [14] introduces genetic programming, an EA that represents solutions as computer programs, arguing that most optimization problems can be reformulated as a problem of finding an optimal program for solving a certain task. The programs are represented in a tree structure, with branches for operations that take more than one argument. Here, the crossover operator usually takes two parents and exchanges random subtrees of the solutions, creating two offspring, while the mutation operator takes a leaf node or subtree and replaces it with a new random leaf or subtree.

3.2 EAs in path-planning

Evolutionary algorithms have been successfully applied to the path-planning problem as a replacement to classical path-planning algorithms in many environments. Real valued, vaiable-length genetic algorithms are the by far most popular choice in path-planning, but there exist other variant, such as the genetic programming algorithm in [11].

The easiest choice of initial population is to generate a set of random paths, feasible or unfeasible. This leads to a long initial planning time, as it may take a while for the algorithm to evolve good, or even just feasible solutions if the environment is complex. [21] suggests using a coarse probabilistic roadmap to generate a set of feasible initial solutions from which to optimize, which will improve planning time in some cases, but this method is still vunerable to very complex environments, since then the PRM would need a large number of nodes in order to find feasible solutions.

3.2.1 Evolutionary Planners in unknown Environments

The real advantage of path-planning using evolutionary algorithms, however, comes when re-planning in an unknown environment. The evolutionary plan-

ner will already have a large collection of other plans availible from the initial planning, some of which might still be feasible. Even if there aren't, it is just a matter of running the algorithm a couple of iterations, re-ranking and evolving the population to fit the new conditions. [27] shows a good example of this, while also illustrating some of the many specialized crossover and mutation operators used in evolutionary path-planning.

Actually, if computing resources allow, there is no reason to ever stop iterating the evolutionary algorithm while the robot is moving, since you cannot know whether the the optimal solution has been found or if the current best solution is only near-optimal. Also, when there are moving obstacles the algorithm can begin adapting to the changes in the environment before they directly block the path the robot is currently following, as shown in [25]. The robot does not even have to wait until the initial planning is entirely done, but can use an intermediate result to guess on a goo initial heading, for example.

4 Multi-objective Optimization

When one wants to find problem solutions that are optimal with relation to not only one, but several objectives one is presented with a multi-objective optimization problem (MOOP). A good introduction to the topic, as seen from a EA perspective, can be found in [6] and [5]. In mathematical terms, instead of a scalar objective function f(**x**) we have a m-dimensional vector objective function

$$\mathrm{F}(\boldsymbol{x}) = \left[\begin{array}{cccc} \mathrm{f}_1(\boldsymbol{x}) & \ldots & \mathrm{f}_{m-1}(\boldsymbol{x}) & \mathrm{f}_m(\boldsymbol{x}) \end{array} \right]^{\mathrm{T}}$$

comprised of m scalar objective functions. In contrast to single-objective optimization problems where there is always only one optimal solution, there may be any number of different solutions in an MOOP that are better according to one objective function and worse according to another in relation to each other and usually there won't be a single solution that is best in every way.

A common example of a MOOP is that of cost versus quality. In a store we may choose between four models of computers. Model A is more powerful than model B, which again is more powerful than model C, which finally is more powerful than the serverly outdated model D. Based only on how powerful the models are, the obvious choice would be model A. But, it so happes that the models range from expensive to cheap in the same order. Then we would have to make a careful consideration of how much computing power we can afford.

4.1 Reduction to single objective

If we had set up a strict budget before going to to the computer store, then our choice would be simple - just buy the most powerful computer below a certain price. Setting up a worst-acceptable value for each objective function except one and the optimizing that single objective within those limits is called the ϵ-constraint method. If do not have such a budget, and find it a bit arbitrary to make up one on the spot, we might think up some some measure of how much value computing power has to us relative to money spent how much weight

is laid on cheapness and performace. We find a fitting numerical weight for both, multiply the cheapness and the performance with their respective weights and sum them up. Now we have a single number signifying how good we feel each computer is for us and we can choose the one that is best. This is caled the weighted sum approach. A number of other methods also exist aiming to reduce multiple objectives into one so that we can find a single best solution using a conventional single-objective optimizing algorithm. A overview of the most common ones can be found for example in chapter 3 of [6].

4.2 Pareto-optimality

The reduction methods mentioned above will find a single best computer for us, but ideally we would like to know what the selection is before planning a budget or deciding our preferences weight-wise. Furthermore, we want to avoid taking model E, which is both more expensive and slower than model C into consideration. With the weighted sum approach model E might seem a better choice than either of models A,B or D. Worse, there might not be any pair of weights that make model B the best solution at all.

To make an informed choice among large number of trade-offs we need the concepts of domination and Pareto-optimality. A solution $\boldsymbol{x}$ is said to dominate a solution $\boldsymbol{y}$, written $\boldsymbol{x} \preceq \boldsymbol{y}$, iff it is as good as or better than $\boldsymbol{y}$ for all objective functions and strictly better for at least one objective function:

$$\boldsymbol{x} \preceq \boldsymbol{y} \;\equiv\; \forall i \,(\mathrm{f}_i(\boldsymbol{x}) \trianglelefteq \mathrm{f}_i(\boldsymbol{y})) \;\wedge\; \exists j \,(\mathrm{f}_j(\boldsymbol{x}) \triangleleft \mathrm{f}_j(\boldsymbol{y}))$$

If the goal is to minimize all objective functions then a solution $\boldsymbol{x}$ is pareto-optimal iff there is no other solution in the entire solution space $\mathcal{S}$ that dominates it.

It may well be that there exists a solution that dominates all other solutions, that is, it is better than all other solution in every way. In that case the Pareto-optimal set will only contain that solution, and the objective functions are not in conflict with each other. If that is the case one would probably be able to optimize the problem using only one of the objectives.

4.3 Multi-objective evolutionary Algorithms

For MOOPs there is one big advantage with using an evolutionary algorithm: it inherently optimizes a large number of solutions. Most other kinds of algorithms work on finding only one solution at a time and can only optimize for a single objective function, forcing the use of some sort of reduction method.

Sometimes specialized optimization algorithms with specialized reduction methods can work out pretty well, as in [23]. But finding a good specialized solution might requre a lot of work, and as allways with reduction methods you have to specify the priority of the different objectives beforehand. And while specifying priorities beforehand may not allways be a disadvantage - a reduction method that optimizes multiple objectives in turn according to priority with is utilized well in [17] - it can be a disadvantage in many situations.

The main difficulty with how to adapt evolutionary algorithms to multi-objective problems is how to select individuals for reproduction and to keep good diversity. The simplest solution to this is to choose different objective reductions for different subpopulations, selecting a number of solutions from each subpopuation for crossover, and then let all the selected solutions cross over at random. However, even with a mutation operator added this can lead to "niching" - that solutions cluster up at the optimums for the different reduction methods. This can be amended to some degree by using random reduction weights or varying them over time, but it is still not very effective.

Another way of solving this problem is to rank solutions by dominance, either by counting how many solutions that dominate each solution or by sorting the solutions into non-dominated sets. The first non-dominated set contains all the solutions that are not dominated by any other solution in the entire population. The next non-dominated set contains all solutions that are only dominated by solutions in the first non-dominated set, the third set contains the solutions only dominated by solutions int the first and second sets and so on. The disadvantage to this is that you need to keep track of who dominates who in the entire population, which has high time complexity.

Ranking solutions by which non-dominance set it is in helps keep variation near the Pareto front since solutions that are close to being Pareto-optimal are more likely to survive, no matter where they are along the front. If the algorithm also makes sure solutions are more likely to survive when they are far from other solutions in objective space then a good, varied seletion of solutions along the Pareto front can be achieved.

References

[1] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, September 1991.

[2] B. Baginski. The z3 method for fast path planning in dynamic environments. In *Proceedings of IASTED Conference on Applications of Control and Robotics*, pages 47–52. Citeseer, 1996.

[3] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1018 –1023 vol.2, 1999.

[4] John Canny and Bruce Donald. Simplified voronoi diagrams. *Discrete & Computational Geometry*, 3:219–236, 1988. 10.1007/BF02187909.

[5] C.A.C. Coello, G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer-Verlag New York Inc, 2007.

[6] K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. Wiley, 2001.

[7] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing (Natural computing series)*. Springer, 2008.

[8] G. Foux, M. Heymann, and A. Bruckstein. Two-dimensional robot navigation among unknown stationary polygonal obstacles. *Robotics and Automation, IEEE Transactions on*, 9(1):96 –102, feb 1993.

[9] T. Fraichard. Trajectory planning in a dynamic workspace: a'state-time space'approach. *Advanced Robotics, 13*, 6(8):75–94, 1999.

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

[11] S. Handley. The genetic planner: The automatic generation of plans for a mobile robot via genetic programming. In *Intelligent Control, 1993., Proceedings of the 1993 IEEE International Symposium on*, pages 190 – 195, aug 1993.

[12] J.H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[13] D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233 –255, 2002.

[14] J.R. Koza. *Genetic programming*. Citeseer, 1992.

[15] J.C. Latombe. *Robot motion planning*. Kluwer international series in engineering and computer science. Kluwer Academic, 1991.

[16] S.M. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006.

[17] C. Leger. *Automated synthesis and optimization of robot configurations: an evolutionary approach*. PhD thesis, Citeseer, 1999.

[18] Hong Liu, Yan Li, He Wen, Jingyan Xia, and Tianguang Chu. Hierarchical roadmap based rapid path planning for high-dof mobile manipulators in complex environments. In *Robotics and Biomimetics (ROBIO), 2009 IEEE International Conference on*, pages 189 –195, dec. 2009.

[19] T. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32:108–120, 1983.

[20] L. Lulu and A. Elnagar. A comparative study between visibility-based roadmap path planning algorithms. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3263 – 3268, aug. 2005.

[21] M. Naderan-Tahan and M.T. Manzuri-Shalmani. Efficient and safe path planning for a mobile robot using genetic algorithm. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 2091 –2097, may 2009.

[22] John Reif and Micha Sharir. Motion planning in the presence of moving obstacles. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 144 –154, oct. 1985.

[23] A.R. Soltani and T. Fernando. A fuzzy based multi-objective path planning of construction sites. *Automation in Construction*, 13(6):717 – 734, 2004.

[24] J.P. van den Berg and M.H. Overmars. Roadmap-based motion planning in dynamic environments. *Robotics, IEEE Transactions on*, 21(5):885 – 897, oct. 2005.

[25] J. Vannoy and Jing Xiao. Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes. *Robotics, IEEE Transactions on*, 24(5):1199 –1212, oct. 2008.

[26] R. Vatcha and Jing Xiao. Perceiving guaranteed continuously collision-free robot trajectories in an unknown and unpredictable environment. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1433 –1438, oct. 2009.

[27] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski. Adaptive evolutionary planner/navigator for mobile robots. *Evolutionary Computation, IEEE Transactions on*, 1(1):18–28, 1997.