PDRRTs: Integrating Graph-Based and Cell-Based Planning

Ananth Ranganathan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
ananth@cc.gatech.edu

Sven Koenig
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
skoenig@usc.edu

***Abstract*— Motion-planning problems can be solved by discretizing the continuous configuration space, for example with graph-based or cell-based techniques. We study rapidly exploring random trees (RRTs) as an example of graph-based techniques and the parti-game method as an example of cell-based techniques. We then propose parti-game directed RRTs (PDRRTs) as a novel technique that combines them. PDRRTs are based on the parti-game method but use RRTs as local controllers rather than the simplistic controllers used by the parti-game method. Our experimental results show that PDRRTs plan faster and solve more motion-planning problems than RRTs and plan faster and with less memory than the parti-game method.**

I. Introduction

Motion planning [1] involves finding trajectories in high-dimensional continuous configuration spaces, for example, by using discrete search methods after discretizing the configuration spaces. Configuration spaces can be discretized in different ways, for example with roadmap or cell-decomposition techniques:

Roadmap techniques [2] [3] [4] [5] determine graphs that lie in freespace and represent its connectivity. Systematic techniques are not well suited for high-dimensional spaces. An example is techniques that construct Voronoi graphs. Consequently, researchers use sampling techniques. An example is rapidly exploring random trees (RRTs) [6], a simple but versatile roadmap technique that builds trees. Sampling techniques are typically probability-complete, meaning that they find a trajectory, if one exists, with a probability that approaches one as their run time increases.

Cell-decomposition techniques [7] [8], on the other hand, decompose the configuration space into cells. They are typically systematic and thus resolution-complete, meaning that they find a trajectory if one exists within the minimum resolution of the decomposition. Uniform terrain discretizations can prevent one from finding a plan if they are too coarse-grained and result in large spaces that cannot be searched efficiently if they are too fine-grained. Consequently, researchers use nonuniform terrain discretizations. An example is the parti-game method [9], a reinforcement-learning method that starts with a coarse terrain discretization and refines it during execution by splitting cells only when and where it is needed (for example, around obstacles).

In this paper, we propose a novel technique that combines the advantages of RRTs and the parti-game method. Our parti-game directed RRTs (PDRRTs) are based on the parti-game method but use RRTs as local controllers. PDRRTs differ from recent work that studied hybrids of two different sampling techniques, such as RRTs and probabilistic roadmaps [10], because they provide a systematic way of improving the performance of RRTs. The main insight of this paper is precisely that the combination of sampling and systematic techniques can result in very powerful motion-planning techniques.

Depending on their parameters, PDRRTs can behave like RRTs, the parti-game method, or a hybrid. Our experimental results show that PDRRTs can plan faster and solve more motion-planning problems than RRTs because the parti-game method directs the searches performed by the RRTs, which allows PDRRTs to solve more motion-planning problems with small passages. Our experimental results also show that PDRRTs can plan faster and with less memory than the parti-game method because RRTs are more capable controllers than the simplistic controllers used by the parti-game method, which allows PDRRTs to split fewer cells than the parti-game method.

The paper is organized as follows. Section II provides a brief description of RRTs and the parti-game method. Section III introduces our PDRRT method. Section IV presents the experimental setup of our comparison of PDRRTs, RRTs and the parti-game method, and Section V presents our results. Section VI describes several possible improvements of basic PDRRTs that extend their applications. Section VII presents related work and Section VIII presents ideas for future work.

II. Background

In this section, we describe both RRTs and the parti-game method in sufficient detail to be able to describe in the following section how to combine them. We describe the parti-game method in greater detail than RRTs because robotics researchers tend to be less familiar with it.

A. RRTs

RRTs [6] build a tree in freespace, starting at the start point. They repeatedly generate a random sample point and then grow the tree by adding an edge of a given length from the vertex on the tree that is closest to the sample point toward the sample point. RRTs can be biased to grow toward the goal point by returning the goal point (instead of a random point) as sample point with small probability. RRTs overcome the problems of earlier roadmap techniques, such as probabilistic roadmaps [3], by biasing their search toward unexplored regions of the freespace. There exist a number of variations of basic RRTs, for example bi-directional versions, that try to connect a tree that is grown from the start point to the goal point with one that is grown in the opposite direction [11].

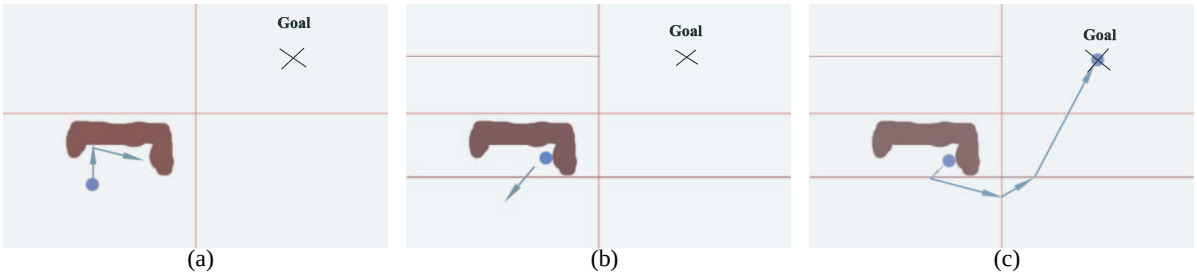
B. The Parti-Game Method

The parti-game method [9] discretizes terrain into rectangular cells of nonuniform size. It starts with an initial coarse terrain discretization and assumes that it has a number of local controllers available in each cell, namely one for each neighboring cell. These controllers must be provided by the user. The parti-game method makes the optimistic default assumption that the execution of each controller from any point in the cell eventually reaches some point in the neighboring cell (with a cost that equals the Euclidean distance between the center of the cells), even though this assumption is not justified since the parti-game method uses a very simplistic controller that just aims for the center of the neighboring cell and can thus, for example, get stuck in front of obstacles. Once the parti-game method has selected a controller, it continues to use the same controller until it either gets blocked by an obstacle or enters a new cell. It uses a minimax search to determine which controller to use in each cell to reach the cell that contains the goal point, under the current assumptions about the effects of the controllers. It does this by determining the minimax goal distance of each cell and assigning it the controller that minimizes its minimax goal distance.

- If the parti-game method finds a solution, it starts to execute it until it either reaches the cell that contains the goal point (it does not need to reach the goal point itself) or the currently used controller has an unexpected effect: it either does not leave the current cell within a given amount of time or it leaves the current cell but reaches a cell different from the intended neighboring cell. If the currently used controller has an unexpected effect, the parti-game method records the newly observed effect as a triple and, from now on, assumes that the the execution of the current controller from any point in the current cell can also result in some point in the cell that the current execution actually resulted in. The observed effects overwrite the default assumption about the effects. The parti-game method then uses another minimax search to again determine which controller to use in each cell under the current assumptions about the effects of the controllers, and repeats the process.
- If the parti-game method does not find a solution, then it assumes that the terrain discretization is too coarse-grained. It therefore refines the terrain discretization by splitting all cells that are unsolvable (that is, have an infinite minimax goal distance), have a size that is larger than the minimum cell size, and border solvable cells - to try to make them solvable. It also splits all cells that are solvable, have a size that is larger than the minimum cell size, and border unsolvable cells - to ensure that neighboring cells have similar sizes. Each cell is split into two cells perpendicular to its longest axis. (The axis of the split is chosen randomly for square cells.) The parti-game method again assumes that it has a number of local controllers available in each new cell, namely one for each neighboring cell. It further makes again the optimistic default assumption that the execution of each controller from any point in the new cell eventually reaches some point in the neighboring cell. This assumption makes the current cell solvable. It then uses another minimax search to again determine which controller to use in each cell under the current assumptions about the effects of the controllers, and repeats the process.

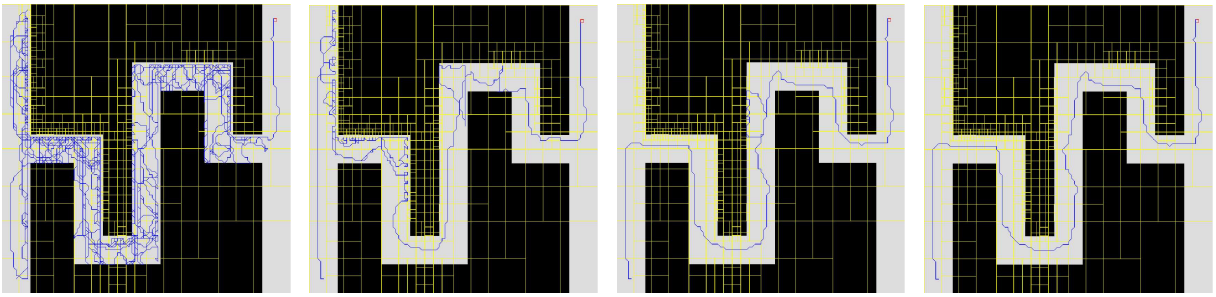
Figure 1 illustrates the behavior of the parti-game method. The circle marks the location of robot and the cross marks the goal region. The robot initially moves up and gets blocked. It then moves right and gets blocked again (a). At this point, the lower-left cell becomes unsolvable. The lower-left cell is now an unsolvable cell that borders solvable cells and the upper-left and lower-right cells are solvable cells that border an unsolvalble cell. Thus, these three cells are split. The robot now moves up and gets blocked immediately. It then moves right and again gets blocked immediately. Finally, it moves down (b) and then succeeds in moving to the goal point (c).

The parti-game method can also be used as a multi-query planner by maintaining the terrain discretization between queries. If it repeatedly solves the same motion-planning problem, for example, then it refines its terrain discretization over time until it converges to both a terrain discretization and a trajectory. All future queries then return this trajectory. Figure 2 shows an example.



(a) (b) (c)

Fig. 1. Example behavior of the parti-game method



Query 1. No. of cells: 574. Trajectory length: 387. (a)
Query 2. No. of cells: 574. Trajectory length: 62. (b)
Query 4. No. of cells: 574. Trajectory length: 52. (c)
Query 6. No. of cells: 574. Trajectory length: 44 (d)

Fig. 2. The behavior of the parti-game method for a sequence of identical motion-planning queries for a zero-link (point) robot in a terrain of size 100×100 with a step size of the local controller of one and a minimum cell size of three.

III. Parti-game Directed RRTs

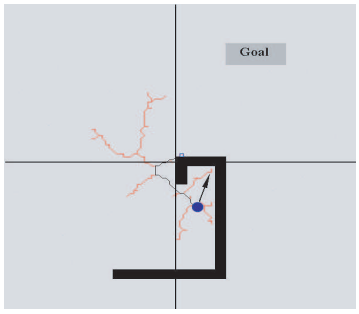


Fig. 3. Example where PDRRTs find a trajectory to the intended neighboring cell even if the parti-game method fails

We now describe a novel technique that combines the advantages of RRTs and the parti-game method. Our parti-game directed RRTs (PDRRTs) are based on the parti-game method but use RRTs as local controllers. PDRRTs can potentially plan faster and solve more motion-planning problems than RRTs because the parti-game method directs the searches performed by the RRTs, which allows PDRRTs to solve more problems than RRTs in terrain with low ε-goodness [12], where ε is the minimum fraction of space visible over all points. An example is terrain with small passages. PDRRTs can also potentially plan faster and with less memory than the parti-game method because RRTs are more capable controllers than the simplistic controllers used by the parti-game method, which allows PDRRTs to split fewer cells than the parti-game method. The controllers of the parti-game method just aim for the center of the intended neighboring cell and can thus easily get stuck in front of obstacles even if the neighboring cell can be reached. This can make the current cell unsolvable and thus result in the parti-game method splitting cells. Figure 3 shows an example where the simplistic controllers of the parti-game method get stuck but RRTs easily find a trajectory to the intended neighboring cell, namely the upper-right cell.

PDRRTs use RRTs as local controllers in the following way after the parti-game method has determined which neighboring cell to move to: The start point is the current point and the goal point is the center of the desired neighboring cell. We impose a limit on the number of nodes in the RRT (we use 250) to limit the search time. In addition, we could impose a bounding box beyond which the RRT cannot grow although we did not do this in our experiments. Whenever our RRTs add a node to the tree that belongs to the intended neighboring cell, then the search terminates and they return the trajectory to that node. If the RRTs fail to

find a trajectory to the desired neighboring cell within the limit on the number of nodes, they could just return failure, similar to the case when the simplistic controllers of the parti-game method get stuck in front of obstacles and thus fail to leave the current cell. However, we found that we can reduce the run time of PDRRTs if our RRTs return a trajectory to that node in the tree that belongs to a different neighboring cell and whose Euclidean distance to the goal point is minimal, similar to the case when the simplistic controllers of the parti-game method leave the current cell but reach a cell that is different from the intended neighboring cell. Our RRTs therefore return failure only if all nodes in the tree belong to the current cell when it reaches the limit on the number of nodes.

If the limit on the number of nodes is small, then the RRTs need to strongly bias their search toward the goal point to have a chance to find a trajectory to it. On the other hand, if the limit on the number of nodes is large, then they should not strongly bias their search toward the goal point to avoid getting stuck in local minima. We therefore bias our RRTs to grow toward the goal point by returning the goal point (instead of a random point) as sample point with the following probabilities that depend on the limit N on the number of nodes:

$$P = \begin{cases} P_{max} & \text{if } N < N_{min} \\ \frac{P_{max} - P_{min}}{N_{max} - N_{min}}(N - N_{min}) + P_{min} & \text{if } N_{min} < N < N_{max} \\ P_{min} & \text{if } N > N_{max} \end{cases}$$

where P_{max}, P_{min}, N_{max} and N_{min} are parameters with $P_{max} > P_{min}$ and $N_{max} > N_{min}$. We use $P_{max} = 1.00$, $P_{min} = 0.05$, $N_{max} = 200$ and $N_{min} = 50$. To summarize, the larger the limit on the number of nodes, the less greedy and thus smarter the local controllers. Depending on both this value and the minimum cell size, PDRRTs can behave like RRTs, the parti-game method, or a hybrid. PDRRTs behave like RRTs if both parameters have large values and like the parti-game method if both parameters have small values. This is an advantage of PDRRTs because it allows them to behave more like RRTs for easy motion-planning problems and more like the parti-game method for harder motion-planning problems.

IV. Experimental Setup and Implementation

In order to compare PDRRTs, RRTs and the parti-game method, we used motion-planning problems for planar articulated robots [13]. An n-dimensional planar articulated robot has n revolute joints and operates in $n+2$-dimensional configuration space (one dimension for each of the joint angles and two dimensions for the x and y coordinates of one of the joints). Thus, when we refer in the following to a 5D motion-planning problem, we mean one with a three-link robot in planar terrain. Each link is of the same length and thus the robots get longer as the number of dimensions increases. The kinematic constraints were given by a limited angular range for each joint and the need to avoid self-collisions. No dynamic constraints were enforced. The robots operated in different kinds of terrain, shown in Figure 4. Terrain (a) is a modification of the one from [14] and requires sudden changes in direction, especially for higher dimensional cases. Terrain (b) and (c) are from [15] and have narrow passages. Terrain (d) is from [16] and somewhat easier than the other ones but, like the other ones, still has a small ε-goodness. The goal region of each terrain was specified by ranges of allowable joint angles and ranges for the x and y coordinates. To measure the distance between two points in the configuration space we used the weighted sum of the squared differences of their $n+2$ coordinates ("weighted Euclidean distance"). Our weights scaled the squared difference of each pair of coordinates to range from zero to one. For example, the weights of all joint angles were $1/(2\pi)$. We used these particular motion-planning problems because they allow us to test the scaling of the various motion-planning methods with respect to the dimensionality of the configuration space without changing the basic nature of the motion-planning problems or losing generality.

Our implementation of RRTs uses the RRTExt method with uni-directional search [11], similar to [17]. (It cannot use the RRTExt method with bi-directional search since our motion-planning problems have goal regions rather than goal points.) It uses kd-trees to efficiently find the vertex on the tree that is closest to the sample point [18]. The RRTs were biased to grow toward the goal point by returning the goal point (instead of a random point) as sample point with probability 0.05.

Our implementation of the parti-game method uses kd-trees to find the neighbors of a cell efficiently. It uses an efficient single-pass method to calculate the minimax goal distances [19]. The local controllers are implemented as follows: There are two actions for each dimension available, which increase or decrease its value by one step size. The parti-game method selects that action that reduces the weighted Euclidean distance to the center of the intended neighboring cell the most. It then selects this action repeatedly as long as it continues to reduce the weighted Euclidean distance. It selects a new action and repeats the process when it no longer reduces the weighted Euclidean distance. It returns when the current cell is exited or a given amount of time has passed. This way of selecting actions resuled in a better trajectory quality than other action-selection strategies that we experimented with.

Finally, our implementation of PDRRTs re-used our

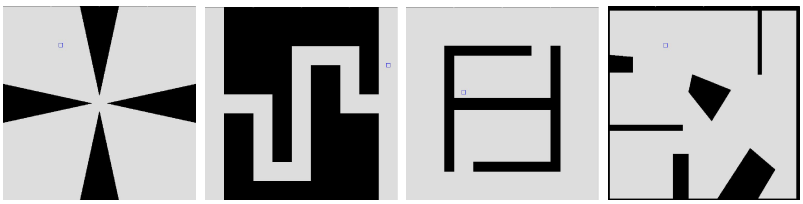


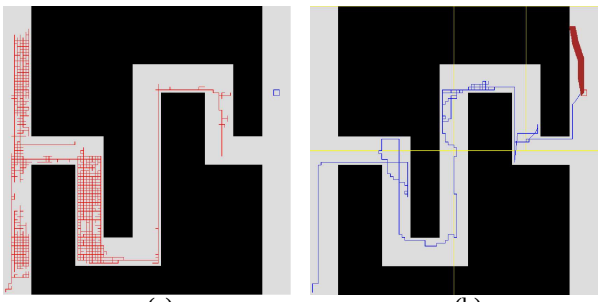
Fig. 4. Test terrains (a)-(d)

implementations of RRTs and the parti-game method whenever possible.

V. Experiments and Results

We evaluate PDRRTs, RRTs and the parti-game method according to how many motion-planning problems they can solve and, for the ones that they can solve, how long it takes and how good the quality of the resulting trajectory is. We used a step size of two for the RRTs and a minimum cell size of five for the parti-game method. Table 8 shows the run times of the three motion-planning methods in our test terrains (a dash indicates that at least one of the motion-planning problems could not be solved within the cut-off time of 60 CPU minutes), Table 10 shows the lengths of the resulting trajectories, and Table 9 shows the number of cells generated by PDRRTs and the parti-game method, in all cases averaged over all motion-planning problems that they solved in 20 runs within the cut-off time.

A. Solvability



(a) (b)

Fig. 5. Solvability example: (a) RRTs and (b) PDRRTs

Earlier, we have hypothesized that PDRRTs are able to solve more motion-planning problems than RRTs. Indeed PDRRTs solved more motion-planning problems within the cut-off time than RRTs and the parti-game method, for example problems with four-link robots in terrain (c) and, as shown in Figure 5, four-link robots in terrain (b). For the latter motion-planning problems, PDRRTs generate 35 cells in 3514.00 seconds before they find a trajectory. PDRRTs appear to be at least as good as RRTs and the parti-game method. (Note

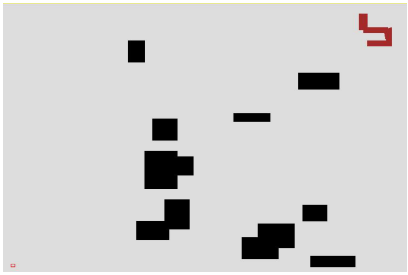


Fig. 6. Randomly generated terrain with a four-link robot at the start location (upper-right corner) and a small square at the goal location (lower-left corner)

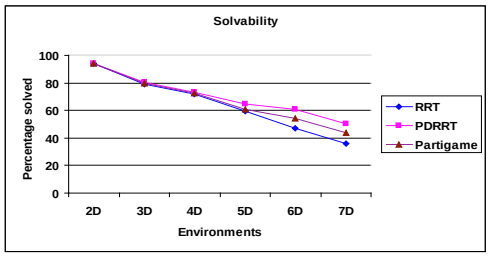


Fig. 7. Number of motion-planning problems solved in random terrain

that we formulated this statement carefully because the run time of PDRRTs was close to the cut-off time in some cases where PDRRTs solved all motion-planning problems but the other methods did not.) To be able to quantify the advantage of PDRRTs over RRTs more precisely, we tested all three motion-planning methods also with zero- to five-link robots in 500 planar terrains that were obtained by randomly generating and placing between 8 to 16 rectangular obstacles into an empty terrain of size 100 by 100, resulting in 2D to 7D motion planning problems. Figure 6 shows an example. We changed both the step size and the minimum cell size to one for this experiment as it makes the largest number of motion-planning problems solvable. Figure 7 shows that PDRRTs solve more motion-planning problems within the cut-off time than RRTs. For example, they solve 20 percent more motion-planning problems than RRTs for the five-link robots. In comparison to the parti-game method, PDRRTs seem to possess only marginally

Terrain	Parti-Game Method	RRTs	PDRRTs
(a) 2D	0.010	0.010	0.010
(a) 3D	0.070	1.410	0.360
(a) 4D	0.790	3.530	0.350
(a) 5D	2623.000	—	172.000
(a) 6D	—	—	2231.000
(b) 2D	0.980	0.850	0.150
(b) 3D	0.230	7.500	1.900
(b) 4D	105.000	38.000	9.760
(b) 5D	2171.000	—	24.900
(b) 6D	—	—	3514.000
(c) 2D	0.092	1.240	0.140
(c) 3D	1.370	12.700	0.690
(c) 4D	5.600	71.500	4.120
(c) 5D	2137.000	1483.000	531.000
(c) 6D	—	—	3263.000
(d) 2D	0.000	0.000	0.000
(d) 3D	0.087	0.140	0.190
(d) 4D	0.183	0.960	0.810
(d) 5D	0.453	1.600	6.610
(d) 6D	26.700	40.300	3.300
(d) 7D	302.000	930.000	172.000

Fig. 8. Run times (in seconds)

Terrain	Parti-Game Method	PDRRTs
(a) 2D	41	1
(a) 3D	32	10
(a) 4D	108	18
(b) 2D	574	31
(b) 3D	70	80
(b) 4D	504	276
(c) 2D	103	30
(c) 3D	128	44
(c) 4D	488	80
(d) 2D	26	1
(d) 3D	35	1
(d) 4D	48	6
(d) 5D	568	120
(d) 6D	1648	64
(d) 7D	1936	192

Fig. 9. Cells generated

Terrain	Parti-Game Method	PDRRTs	RRTs	Parti-Game Method (post-processed)	PDRRTs (post-processed)
(a) 2D	128.00	121.00	111.95	78.00	119.00
(a) 3D	717.80	497.90	216.22	523.20	409.40
(a) 4D	8307.00	520.30	341.93	4407.65	497.60
(b) 2D	3992.00	735.00	307.00	749.00	287.00
(b) 3D	2287.90	6991.80	505.00	1369.20	505.40
(b) 4D	156889.00	4264.60	763.00	31421.90	1123.90
(c) 2D	2430.00	1106.00	299.00	286.00	302.00
(c) 3D	12941.00	3746.30	504.00	8045.65	606.30
(c) 4D	26460.00	15374.20	744.00	17056.50	7006.50
(d) 2D	272.00	120.00	58.90	222.00	89.00
(d) 3D	499.80	165.60	203.00	349.70	165.00
(d) 4D	833.70	234.00	329.00	403.70	234.00
(d) 5D	917.70	241.60	417.00	514.40	241.00
(d) 6D	3711.00	1879.70	525.00	2221.10	485.60
(d) 7D	12782.10	6727.36	625.00	12037.50	1183.60

Fig. 10. Trajectory lengths

greater solvability. The difference in the case of the five-link robot is 6 percent which is not significant.

B. Run Time

Table 8 shows that PDRRTs seem to be faster than RRTs and the parti-game method if the dimensionality of the terrain is sufficiently large. This means, for example, that PDRRTs solve more motion-planning problems than RRTs without being slower.

C. Trajectory Quality

Table 10 shows the lengths of the trajectories of the three motion-planning methods. RRTs produce the shortest trajectories, followed by PDRRTs and eventually the parti-game method. The trajectories of PDRRTs and the parti-game method can be improved in two different ways:

- They can be improved with a simple post-processing step that removes cycles since both motion-planning methods can move back and forth while splitting cells. Figure 2, for example, shows such loops and meanders, whose removal can greatly improve the trajectory.
- They can also be improved by letting the motion-planning methods repeatedly solve the same motion-planning problem. In this case, they refine the terrain discretization and the trajectory over time. The quality of the trajectories tends to improve, although the improvement is not monotonic in time. For example, Figure 11 shows how the length of the found trajectory changes over time for a one-link robot in terrain (c). The trajectory lengths are smaller for PDRRTs than they are for the parti-game method. (For comparison purposes, the average trajectory length is 504.00 for RRTs.) They change no longer after 15 iterations for PDRRTs and 40 iterations for the parti-game method if the trajectories are not post-processed, and after 7 interations for PDRRTs and 23 iterations for the parti-game method if the trajectories are post-processed. Thus, they converge earlier for PDRRTs than for the parti-game method.

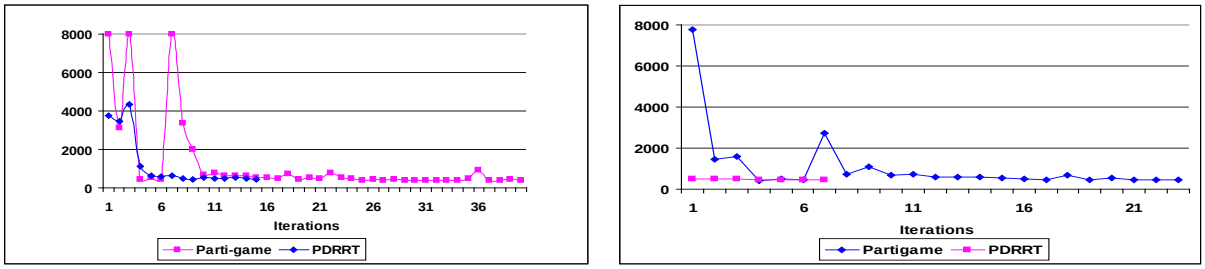
The length of the trajectories for a one-link robot in terrain (c) is 3746.30 for PDRRTs without post-processing. It can be reduced to 606.30 with post-processing and to 402.38 by letting PDRRTs repeatedly solve the same motion-planning problem. It cannot be reduced much further even if both ways are combined. In this case, post-processing is faster than letting PDRRTs repeatedly solve the same motion-planning problem and the quality of the resulting trajectory is only slightly worse.

D. Memory Consumption

In general, it is difficult to compare the memory consumption of PDRRTs, RRTs, and the parti-game method since RRTs and the parti-game method use very different data structures. Table 9 therefore only shows the number of cells generated by PDRRTs and the parti-game method. Earlier, we have hypothesized that PDRRTs generate fewer cells than the parti-game method. This is indeed the case.

VI. PDRRT Extensions

Researchers have investigated various improvements to RRTs and the parti-game method, all of which can be used in the context of PDRRTs as well. Researchers have, for example, investigated versions of RRTs that take repeated steps toward the sample point until an obstacle is encountered, instead of only a single step [11]. They have also investigated versions of the parti-game method that split cells in a different way than the version used by us [19]. There are other ways how one can improve PDRRTs. For example, whenever our RRTs find trajectories to the intended neighboring cells, one can cache them for later reuse. Whenever our RRTs, during their search, find trajectories to neighboring cells that are different from the intended neighboring cells, one can cache them for later use in situations where one actually wants to reach these cells. If one caches the RRTs instead of the trajectories, one can even recover



(a) (b)

Fig. 11. Performance profiles (a) without post-processing (b) with post-processing

trajectories to the neighboring cells after cells have been split. These improvements can potentially reduce the run time of PDRRTs and are especially important for multi-query planning. One can also vary the step size of the RRTs so that they use a small step size if the sizes of the current and intended neighboring cells are small and a larger step size if they are large. The idea behind this suggestion is that small step sizes waste run time in large cells but are necessary to find trajectories in small cells. For example, the step size should not be larger than the cell sizes. A simple heuristic thus is to set the step size to some fraction of the cell sizes.

VII. Related Work

PDRRTs relate to other research directions in motion planning. For example, PDRRTs use more sophisticated local controllers than the parti-game method. The effect of local controllers has been studied in the motion-planning literature in the context of probabilistic roadmaps [20], that repeatedly generate random sample points and then use the local controllers to connect them to the existing roadmap. Thus, roadmaps call the local controllers much more frequently than PDRRTs, which explains why they tend not to use sophisticated and thus slow local controllers. The caching of results from previous queries has been studied in the motion-planning literature in the context of RRTs that maintain the trees between queries [21], but these rapidly-exploring random forests need to prune the trees to be efficient. Thus, PDRRTs have the potential to be faster multi-query planners than rapidly-exploring random forests.

In this paper, we propose a novel technique that combines the advantages of RRTs and the parti-game method. Our parti-game directed RRTs (PDRRTs) are based on the parti-game method but use RRTs as local controllers. PDRRTs differ from recent work that studied hybrids of two different sampling techniques, such as RRTs and probabilistic roadmaps [10], because they provide a systematic and deterministic way of improving the performance of RRTs. The core insight of this paper is that the combination of sampling and systematic techniques can result in very powerful motion-planning techniques.

The motion-planning technique most closely related to PDRRTs uses RRTs as local controllers for probabilistic roadmaps [10]. Both motion-planning techniques can be used as single-query and multi-query planners. Their main difference is that PDRRTs combine a systematic technique with RRTs (a sampling technique), whereas the other motion-planning technique combines another sampling technique with RRTs. Disadvantages of combining two different sampling techniques are that the performance of the resulting motion planner is very sensitive to the choice of parameter values and can completely degrade in difficult environments, as was noted in both [10] and [22]. The performance of PDRRTs, in contrast, is more robust. It is insensitive to the exact parameter values and does not completely degrade in difficult environments because PDRRTs compensate for the failure of RRTs by making additional calls to the systematic technique, resulting in smaller cells and thus additional guidance for the RRTs. Our paper thus demonstrates that the combination of sampling and systematic techniques can result in very powerful motion-planning techniques.

VIII. Future Work

While PDRRTs solve more motion-planning problems than RRTs, they remain only probability-complete because their local controllers can fail to find trajectories between cells even if they exist. It is future work to make them resolution-complete. One could, for example, first use the simplistic (but systematic) controllers of the parti-game method in every cell and, only if they fail, then switch to the (sampling-based) RRTs. One could also use quasi-random number sequences, especially incremental sequences [23], when generating “random” sample points to grow the RRTs. This overcomes the disadvantages of sampling-based methods while maintaining their strengths. It is also future work to perform more extensive experiments with PDRRTs, for example, with different motion-planning problems than the ones that we used here and for the proposed improvements of the basic PDRRT method. This is interesting because the parti-game method has been reported not to work

well for manipulator-trajectory problems [24]. If this is due to the fact that the simplistic local controllers of the parti-game method tend not to reach the intended neighboring cells in the presence of non-holonomic kinematic or dynamic constraints, then PDRRTs with their more sophisticated controllers might work well even for manipulator-trajectory problems. In general, it is important to understand better when the parti-game method as well as motion-planning methods based on the parti-game method, such as PDRRTs, perform well and when they do not.

IX. Conclusions

In this paper, we proposed parti-game directed RRTs (PDRRTs) as a novel technique that combines rapidly exploring random trees (RRTs), a graph-based discretization technique, and the parti-game method, a cell-based discretization technique. PDRRTs are based on the parti-game method but use RRTs as local controllers rather than the simplistic controllers used by the parti-game method. Our experimental results show that PDRRTs plan faster and solve more motion-planning problems than RRTs and plan faster and with less memory than the parti-game method. We also described several possible improvements of basic PDRRTs that extend their applications, including a version that uses RRTs with variable step sizes.

Acknowledgments

We thank Andrew Moore for his support of this work and Ronald Arkin for his valuable comments. This research is supported under DARPA's Mobile Autonomous Robotic Software Program under contract #DASG60-99-C-0081. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

References

[1] J. C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[2] M. Overmars and P. Svestka, "A probabilistic learning approach to motion planning," in *Algorithmic Foundations of Robotics, The 1994 Workshop on the Algorithmic Foundations of Robotics, A. K. Peters*, Goldberg, Halperin, Latombe, and Wilson, Eds., 1995.

[3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration space," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] Y. W. N. Amato, "A randomized roadmap method for path and manipulation planning," in *IEEE International Conference on Robotics and Automation*, 1996, pp. 113–120.

[5] W. Henning, F. Hickman, and H. Choset, "Motion planning for serpentine robots," in *Proceedings of ASCE Space and Robotics*, 1998.

[6] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept, Iowa State University, Tech. Rep. TR 98-11, October 1998.

[7] S. Kambhampati and L. Davis, "Multiresolution path planning for mobile robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 3, pp. 135–145, 1985.

[8] H. Noborio, T. Naniwa, and S. Arimoto, "A quadtree-based path-planning algorithm for a mobile robot," *Journal of Robotic Systems*, vol. 7, no. 4, pp. 555–574, 1990.

[9] A. W. Moore, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," in *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6. Morgan Kaufmann Publishers, Inc., 1994, pp. 711–718.

[10] M. Akinc, K. Bekris, B. Chen, A. Ladd, E. Plakue, and L. Kavraki, "Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps," in *The Eleventh International Symposium on Robotics Research*, 2003.

[11] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *IEEE Intl. Conf. on Robotics and Automation*, 2000, pp. 995–1001.

[12] L. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan, "Randomized query processing in robot motion planning," in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, 1995, pp. 353–362.

[13] E. Sacks, "Path planning for planar articulated robots using configuration spaces and compliant motion," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 381–390, 2003.

[14] V. Boor, M. Overmars, and A. van der Stappen, "The gaussian sampling strategy for probabilistic roadmap planners," in *IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1018–1023.

[15] J. Kuffner, S. Kagami, M. Inaba, and H. Inoue, "Performance benchmarks for path planning in high dimensions," in *Proc. 2001 JSME Conf. on Robotics and Mechatronics (ROBOMEC'01)*, Takamatsu, Japan, June 2001.

[16] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4/5, pp. 495–512, 1999.

[17] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *Proceedings of IROS-2002*, 2002.

[18] A. W. Moore, "An introductory tutorial on kd-trees," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. Technical Report No. 209, Computer Laboratory, University of Cambridge, 1991.

[19] M. Al-Ansari, "Efficient reinforcement learning in continuous environments," Ph.D. dissertation, College of Computer Science, Northeastern University, 2001.

[20] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo, "Choosing good distance metrics and local planners for probabilisitc roadmap methods," in *IEEE International Conference on Robotics and Automation*, 1998, pp. 630–637.

[21] T.-Y. Li and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management," in *IEEE International Conference on Robotics and Automation*, vol. 4, 2002, pp. 3411–3416.

[22] S. LaValle and J. Kuffner, "Rapidly exploring random trees: Progress and prospects," in *Workshop on the Algorithmic Foundations of Robotics*, B. Donald, K. Lynch, and D.Rus, Eds. A.K. Peters, 2000, pp. 293–308.

[23] S. R. Lindemann and S. M. LaValle, "Incremental low-discrepancy lattice methods for motion planning," in *IEEE International Conference on Robotics and Automation*, 2003.

[24] M. Eldracher and R. Merklein, "Why the parti-game algorithm does not work satisfyingly for manipulator trajectory generation," Computer Science Department, Technical University Munich, Tech. Rep., unknown year (unpublished).