Multi-agent RRT*: Sampling-based Cooperative Pathfinding

Michal Čáp[1], Peter Novák[2], Jiří Vokřínek[1], and Michal Pěchouček[1]

[1] Agent Technology Center, DCSE, FEE, Czech Technical University in Prague
[2] EEMCS, Delft University of Technology, the Netherlands

Abstract. Cooperative pathfinding is a problem of finding a set of non-conflicting trajectories for a number of mobile agents. Its applications include planning for teams of mobile robots, such as autonomous aircrafts, cars, or underwater vehicles. The state-of-the-art algorithms for cooperative pathfinding for embodied robots typically rely on some heuristic forward-search pathfinding technique, where A* is often the algorithm of choice. Here, we propose MA-RRT*, a novel algorithm for multi-agent motion planning that builds upon a recently proposed asymptotically-optimal sampling-based algorithm called RRT*. In this paper, we focus on the case where the agents' mobility model is a discrete graph. We evaluate the performance of the proposed algorithm and its scalability with respect to the number of agents and the size of the environment. Our results show that the sampling-based approach offers better scalability than the classical forward-search approach in relatively sparse environments, which are typical in real-world applications such as multi-aircraft collision avoidance.

1 Introduction

Consider a group of mobile robots, such as autonomous aerial, ground, or underwater vehicles operating in a shared space. One of the fundamental issues in scenarios involving such agents is the problem of planning trajectories for the individual agents so as to avoid collisions between them.

The problem of collision avoidance for mobile robots, such as aircrafts can be modeled as an instance of *cooperative pathfinding*, a relatively well studied problem of finding a set of non-conflicting trajectories for a number of mobile agents. The general setting for cooperative pathfinding problem involves an environment shared by a number of agents defined by its starting and its destination positions. Optionally, the environment can be structured and contain obstacles, possibly the problem description can involve also a model of the agents' physical dynamics, in the case of physical embodied agents. Traditionally, cooperative pathfinding has been modeled in highly structured environments, such as grids either including a relatively large number of obstacles, and/or with a relatively high number of agents. The problem of cooperative pathfinding is known to be PSPACE-hard [4].

Modeling collision avoidance for vehicles, such as autonomous aircrafts, or underwater vehicles, as an instance of cooperative pathfinding leads to instances with a relatively small number of agents operating in relatively large environments involving

relatively few, however possibly large, obstacles. In general, while focusing on the combinatorially intensive instances of cooperative pathfinding it is important to ensure theoretical properties, such as completeness, or optimality. The straightforward complete approach to the problem is to search the solution in what we call a joint-state space. Such a state space is constructed as the Cartesian product of the state spaces of the individual agents. A state in joint-state space is a tuple of the positions of individual agents. A transition between two states in such a space are done in terms of joint-actions, which are again tuples of actions of individual agents. The solution is a joint plan for the agent team, so that all the agents reach their final destinations by a series of valid movements without colliding with each other. Such a space is typically searched using some heuristic forward search algorithm, such as A* [3]. The performance of forward search algorithms hinges on low branching factor of the search space, which is in joint-action spaces often exponential in the number of agents. Suppose for example that an agent can move in four directions and that the problem involves six agents. Then, there is $4^6 = 4096$ possible joint actions at each timestep! One can see that the completeness of such an approach is traded off for a prohibitive computational cost.

Recently, Karaman and Frazzoli [7] introduced a novel sampling-based motion planning algorithm that offers a good scalability to large high-dimensional environments, while at the same time it guarantees asymptotic convergence to an optimal solution. The approach typically discovers an any-quality solution relatively fast and spends the remaining time on improving the solution. In solving collision avoidance for multi-robot teams, while still important, optimality is not of the primary concern. Rather, it is the speed of planning which is more important, even if the cost should be a slight decrease in solution quality.

The main contribution of this paper is MA-RRT* – a sampling-based algorithm for cooperative pathfinding. At the core, the algorithm marries two approaches: while relying on planning of agents' movements in their joint-state space, it replaces the A*-based heuristic search in the joint-state space by RRT*, a fast sampling-based algorithm (Section 4). In Section 5 we extensively evaluate the performance, scalability, as well as the quality of solution produced by the algorithm and show that for sparsely populated large environments the sampling algorithm outperforms Standley and Korf's optimal anytime algorithm (OA) in terms of runtime and success rate, while still maintaining reasonable quality of the solution. We conclude the discourse in this paper by a discussion of the algorithm's limitations, as well with final remarks on ongoing and future work towards its promising extensions.

2 Related Work

In 2010, Standley [12] introduced an optimal and complete cooperative pathfinding solver based on A* heuristic search coupled with two efficient search strategies: (i) independence detection (ID) and (ii) operator decomposition (OD). The ID strategy starts by identifying subproblems that can be solved independently without harming the optimality of the solution. First, a path is planned for each agent independently. Then, the resulting paths are checked for mutual conflicts. Non-conflicting paths are kept as a part of the global solution, the conflicting paths are joined to groups and solved

in the agents' joint-state space. The joint-state space is searched using the operator decomposition technique that decomposes joint-actions to trees of single-agent actions, which allows more efficient pruning during the search process. The OD+ID technique was further extended to an optimal anytime algorithm (OA) by Standley and Korf in 2011 [13]. In OA algorithm, a path is found for each agent independently. If the conflict between two paths is detected, the algorithm tries to avoid joining the paths into a group by generating an alternative path for one of the agents that avoids conflict with the other agent, who is represented as a moving obstacle for the path planner. Usually, such alternative paths can be found, but they are often suboptimal. When the first feasible solution is found and returned, the algorithm uses the remaining time to improve the solution. The algorithm takes the paths that previously conflicted (and thus may be suboptimal), joins them into a group and searches for optimal multi-agent solution in joint-state space consisting of all agents in the new group. The OA algorithm has been originally defined for agents moving on a grid., It can be trivially extended to agents moving on graphs as long as the motions of individual agents have identical durations.

Besides the above presented approaches based on optimal forward search, there is also a class of methods for cooperative pathfinding, such as Push and Swap [9] or BIBOX [14], that do not target the quality of the solution, but attempt to constructs any feasible multi-agent plan.

In the recent two decades, incremental sampling based techniques for motion planning gained popularity for its ability to quickly find solutions to hard high-dimensional motion-planning problems. Probably the most well-known class of sampling-based algorithms are the rapidly exploring random trees (RRT) [8]. The main principle behind the algorithm is incremental construction of a tree of states that starts at the start state and grows towards random samples taken from the state space. When a new random state is sampled, the algorithm finds the nearest state (with respect to some distance metric) that is already in the tree and extends the tree from that state to the random sample. The algorithm has been proved to be probabilistically complete, i.e. the probability that the algorithm fails to find a solution exponentially decays to zero with increasing number of samples. The main drawback of the algorithm is that it does not pay any attention to quality of the solution.

An important progress in the field of sampling-based planning was made in 2011, when Karaman and Frazzoli [7] published the RRT* algorithm, an anytime extension of RRT that converges to optimal solutions. Unlike RRT, RRT* does not connect the random sample to the nearest state already in the tree, but instead it considers all states from the tree that lie in a ball of certain radius centered at the random sample. From these "near" states, it connects the random sample to a near state that leads to to the lowest-cost path from the start state. Building on the random graph theory, the authors of the algorithm were able to prove that if the ball is above certain volume, the algorithm (almost surely, under some mild technical assumptions [7]) converges to a tree that contains optimal paths to all points in the space. Unlike search-based path planning methods that search for a solution that is optimal with respect to the chosen discretization of the state space, RRT* converges to a continuously optimal solution, which makes both approaches directly incomparable.

There have been attempts to use RRTs for search in joint-state space of a number of robots for multi-robot path planning. Most of the work is based on non-optimal RRTs and therefore these approaches cannot offer convergence to an optimal solution, e.g. [6]. The closest work to our approach is the search in multi-robot joint-state space using Anytime RRT technique, which is based on a series of repeated RRT runs, where the maximum cost of each search is bounded by the cost of solution from the previous run [10, 2]. None of these works, however, attempts to make a comparison to the state-of-the art techniques developed in the field of cooperative pathfinding as we do here.

The forward-search techniques applied to motion planning problems search for an optimal path on [2, 10] a discretization of the state space given in form of a graph. One of the popular discretization techniques for continuous configuration spaces are state lattices proposed by Pivtoraiko et al. [11]. Such a discretization is commonly used to model dynamics and find feasible paths in wide variety of real-world motion planning problems including planning for autonomous cars (e.g. DARPA Urban Challenge winner in 2007 [1]) or UAVs [5]. The main advantage of the state lattice representation is that it is regular and thus it can be made implicit, i.e. the nodes and edges of the graph can be computed on the fly only when they are requested. A specific instance of a state lattice is a motion graph $G^M = (W, M)$ that discretizes given Euclidean space into a finite set of spatial waypoints W and a finite set of motion primitives M that represent feasible motions agents may execute to move between two waypoints in W. In the following we show the performance of our technique for multi-agent pathfinding on motion graphs.

3 Problem Formulation

To allow fair comparison with the OA algorithm we define the cooperative pathfinding problem as follows. Consider n agents labeled $1, \ldots, n$ operating in an Euclidean space. The motion model of the agent i is described by a corresponding motion graph denoted as $G_i^M = (W_i, M_i)$. For simplicity, we assume that all motion primitives have the same duration, i.e. $\forall i \in \{1, \ldots, n\} : \forall m \in M_i : dur(m) = c$, where $dur(m)$ is the time-duration of a motion primitive m and c is a constant. The starting positions of all agents are given as an n-tuple $(s_1, \ldots, s_n)$, where $s_i \in W_i$ is the starting waypoint of agent i. Similarly, $(d_1, \ldots, d_n)$ is an n-tuple of destination waypoints $d_i \in W_i$ of each agent. The task is to find a sequence of motion primitives, i.e. a path p_i in the motion graph G_i^M for each agent i, such that $startvertex(p_i) = s_i$ and $endvertex(p_i) = d_i$ and the paths are separated, i.e. $\forall j, k : j \neq k \Rightarrow sep(p_j, p_k)$, where $sep(p_i, p_j)$ denotes a space-time separation relation between paths p_i and p_j. We say that trajectories are separated with respect to the given separation distance d_{sep} if and only if $d^E(p_i[t'], p_j[t']) > d_{sep}$ for all time points t', where $d^E(\cdot, \cdot)$ is the Euclidean distance. The solution to the problem is therefore an n-tuple of paths $(p_1, \ldots, p_n)$. As the solution quality metric we use the sum of times each of the agents spends outside his destination waypoint. Formally,

$$cost(\mathbf{p}) = \sum_{i=1}^{n} \sum_{m \in p_i} \begin{cases} 0 & \text{if } s(m) = d(m) = d_i \\ dur(m) & \text{otherwise} \end{cases},$$

where $s(m)$, $d(m)$ and $dur(m)$ denote the start vertex, the destination vertex and the time-duration of the motion primitive m. Such a metric can account for situations in which an agent must leave his destination to pass through another agent. Further, it was used by Standley [12, 13] for evaluation of ID, OD and OA algorithms. We adopt the metric as well to simplify the comparison with other state-of-the-art methods.

4 Sampling-based approach

Our approach to cooperative pathfinding builds upon a novel sampling-based motion planning algorithm called RRT* [7]. The algorithm is designed for continuous state spaces in which it can efficiently find a path from a given start state to a given goal region[3] by incrementally building a tree that is rooted at the start state and spans towards randomly sampled states from some given state space. Once such a tree first reaches the goal region, we can follow its edges backwards to obtain the first feasible path from start state to the target region. However, even after the first solution is returned, the algorithm does not stop, but instead it continues extending the tree by drawing new random samples, which leads to incremental discovery of new lower-cost paths. Before exposing the details of the algorithm, we have to provide definitions of the following primitive procedures. In the definitions we assume that the vertices in the graphs are states from some given state space C and the distance function $dist(x, y)$ is a function that returns some lower bound on the cost of a path between x, y, where $x, y \in C$.

Nearest Neighbor: Given a graph $T = (V, E)$ and a state $x \in C$, the function $Nearest(T, x)$ returns a vertex $v \in V$ that is the closest to state x in terms of the given distance metric.

Near Vertices: Given a graph $T = (V, E)$, a state $x \in C$ and the numbers $n, m \in \mathbb{N}$, the function $Near(T, x, n, m)$ returns a set of all vertices within a closed ball of radius r_n centered at x, where $r_n = \max\left\{\gamma(\log n\,/n)^{1/d},\, m\right\}$, γ is a constant and d is the dimensionality of the space C.

Local Steering Procedure: Given a graph $T = (V, E)$ and two states x and y the domain-specific local steering procedure determines whether and how x can be extended towards y. The result is a point x_{new} that is closer to y than x is and trajectory/path in the state space from x to x_{new}. Note that it is possible that $x_{new} \neq y$.

The main loop of the RRT* algorithm is exposed in Algorithm 1. Until interrupted, the algorithm samples states from the state space and uses the drawn random sample to extend the tree. .

The $Extend(T, x_{rand})$ routine consists of three main steps. First, it finds the nearest neighbor $x_{nearest} = Nearest(T, x_{rand})$. Then, it tries to extend the tree from $x_{nearest}$ towards x_{rand} using the local steering procedure. The result of steering is a new point in the state space denoted as x_{new}. Then, a set of of vertices X_{near} that are near the x_{new} state is constructed by calling the $Near(T, x_{new}, |V|, m)$ procedure. The X_{near} set is constructed for two purpose. Firstly , the algorithm examines all vertices in X_{near} and picks the vertex from X_{near} that yields the lowest cost if it is chosen as a parent. After that, the new vertex is added to the tree as a child of the chosen parent. The final step is

[3] Sampling-based algorithms work with the concept of goal region instead of the goal as a single point.

Algorithm 1 RRT*

1: $V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$;
2: **while** not interrupted **do**
3: $T \leftarrow (V, E)$;
4: $x_{rand} \leftarrow$ SAMPLE
5: $(V, E) \leftarrow$ EXTEND(T, x_{rand})
6: **end while**

rewiring. The algorithm examines all vertices from X_{near} to see whether their cost can be improved by going through the new vertex. If there are any such vertices, the tree is rewired so that these vertices become children of the new vertex, which allows them to improve the cost of their path from the start state.

4.1 Graph version of RRT*

Original formulation of RRT* is designed for continuous configuration space. In this section we introduce a modified graph version of RRT* (G-RRT*) that works on a state space that is discretized in form of a motion graph. The motivation for this is twofold. First, the G-RRT* will allow us to compare performance of our multi-agent algorithm with the other state-of-the-art methods on the same problem instances. Second, (as a by-product) it allows us to show the applicability of sampling-based planning to graph search problems.

The main loop of the G-RRT* algorithm is identical to the original RRT* with the following modifications. The starting state x_{init} is the start waypoint s. The target region is only the vertex d. The EXTEND procedure is shown in Algorithm 2. The core difference is in the implementation of the steering procedure, which is in G-RRT* done using heuristic-guided greedy search in the motion graph as shown in Algorithm 3, where $h(\cdot)$ is a heuristic function guiding the greedy search and the parameter c_{max} is a user-specified threshold after which is the search terminated. This way, G-RRT* incrementally constructs a tree that is rooted at the start vertex and spans to the randomly sampled vertices from the motion graph. Then, the edges in such a tree represent paths in the motion graph G^M.

Let $G^M = (W, M)$ be a connected motion graph and $s, d \in W$ be starting and destination waypoints in G^M respectively. We also assume SAMPLE samples every vertex of G^M with a non-zero probability and $h(\cdot)$ is a metric, hence it satisfies the triangle inequality. Further, we assume that the m parameter of NEAR procedure that bounds the minimum radius of the near-ball is greater or equal than the length of the longest edge in G^M.

Lemma 1. *The algorithm G-RRT* with arguments G^M, s and d will eventually generate a tree T containing all the vertices from G^M and a path from s to d in T that corresponds to an optimal route from s to d in G^M.*

Proof. (sketch) Note, the edges in the constructed graph T represent paths between nodes in the motion graph G^M. That is, the existence of the edge (x_1, x_2) in T means,

Algorithm 2 EXTEND(T, x)

$V' \leftarrow V$; $E' \leftarrow E$
$x_{nearest} \leftarrow$ NEAREST(T, x)
$(x_{new}, p_{new}) \leftarrow$ GREEDY($G_M, x_{nearest}, x$)
if $p_{new} \neq \emptyset$ **then**
    $V' \leftarrow V' \cup \{x_{new}\}$
    $x_{min} \leftarrow x_{nearest}$
    $X_{near} \leftarrow$ NEAR($T, x_{new}, |V|$)
    for all $x_{near} \in X_{near}$ **do**
        $(x', p') \leftarrow$ GREEDY(G_M, x_{near}, x_{new})
        if $x' = x_{new}$ **then**
            $c' \leftarrow cost(x_{near}) + cost(x_{near}, x_{new})$
            if $c' < cost(x_{new})$ **then**
                $x_{min} \leftarrow x_{near}$
            end if
        end if
    end for
    $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
    for all $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
        $(x'', p'') \leftarrow$ GREEDY(G_M, x, x_{near})
        if $x'' = x_{near}$ and $cost(x_{near}) > cost(x_{new}) + cost(x_{new}, x_{near})$ **then**
            $x_{parent} \leftarrow parent(x_{near})$
            $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
        end if
    end for
    return $G' = (V', E')$
end if

Algorithm 3 GREEDY(G_M, s, d)

procedure GREEDY(G_M, s, d)
    $x \leftarrow s$; $c \leftarrow 0$; $path \leftarrow \emptyset$
    while $x \neq d$ and $c \leq c_{max}$ **do**
        $x' \leftarrow \arg\min_{x \in children(G_M, x)} h(x)$
        $c = c + cost(x, x')$; $path = path \cup (x, x')$; $x = x'$
    end while
    return $(x, path)$
end procedure

that it is possible to reach x_2 from x_1 by simply relying on greedy search based on the Euclidean distance $h(\cdot)$ to the target vertex.

convergence: the sampling function is fair in that in an infinite time, it would select each vertex of G^M infinitely often. The idea underlying the EXTEND routine is that it tries to extend the graph G either with a new randomly sampled vertex if such can be reached from the nearest vertex in T by greedy search, or some other vertex, such that there exists a greedy path from the nearest vertex in T to that vertex. Realize however, that since the graph G^M is connected, it is always possible to extend it with some vertex of G^M which is a neighbor of some existing vertex of T and since there exists a path between the vertex s and every other vertex of G^M, there also exists a sequence of vertices which can be pairwise connected by greedy search so that eventually the last vertex on that path can be reached. The paths are finite, hence the probability that the fair sampling routine eventually samples all the vertices along every finite path is non-zero, and in turn eventually all the vertices of G^M get included in T at which moment the algorithm converges to the point when it contains paths to all vertices in G^M.

soundness: first realize, that at every moment, the graph T is a tree rooted in s. This is ensured by the fact that s is the first vertex with which the construction of T is initialized and then by the way how the new edges are added to the graph. It is always a new vertex, without any child vertices, that is connected to a near vertex already present in T, its new parent. Furthermore, whenever some vertices are rewired, they are first disconnected from their old parents and only then connected to new ones. This way, every vertex has always exactly one single parent, hence at every moment T is a tree. Secondly, see that every path of the tree T rooted in s corresponds to a valid path in G^M. Given a path $s = x_0, \ldots, x_n$ in T, the corresponding path in G^M can be obtained by joining all the vertex sequences resulting from a greedy search between each pair (x_i, x_{i+1}). In a consequence, at any moment when the destination vertex d is already included in T, the path from s to d is a valid traversable path in G^M.

completeness: we need to show that whenever a path between s and d exists in G^M, the algorithm G-RRT* will find it. This straightforwardly follows from the proofs of the algorithm's convergence and soundness. Eventually, T will be extended with d and from that moment on, there will exist a path in T that corresponds to some path from s to d in G^M.

optimality: denote the optimal path between the vertices s and d on graph G^M as $(s = v_1, \ldots, v_k = d)$. The fact that the radius of the near search r_n is bigger than the longest edge in graph G^M implies that if a vertex is sampled, all his neighbors on the G^M graph will be in the X_{near} set. We can recall that the NEAR procedure operates on the contents of X_{near}. It connects a vertex v to the best parent from X_{near} and it rewires other vertices to the vertex v if it leads to improvement of their cost. From fair sampling assumption we know that each of the vertices on the optimal path will be sampled infinitely many times. Eventually, the start vertex v_1 will be sampled, which will lead to rewiring of the vertex v_2 so that it is connected to v_1. This path ending in v_2 is an optimal one and as such it won't be changed during subsequent sampling. After the vertex v_2 has been fixed at optimum cost, the sampling procedure will eventually pick vertex v_3 that will necessarily connect to v_2 and rewires v_4. We can continue by induction up to the vertex v_k, which is the last vertex of the optimal path. □

When implementing G-RRT*, one has to be careful to limit the shrinkage of the near-ball during the sampling. If the radius of the ball gets smaller than the length of the shortest edge in the graph, then the X_{near} set will be always empty and the algorithm will stop optimizing the paths. On the other hand, the value of r_n never needs to be larger than c_{max} parameter of the GREEDY procedure, since states that are further away than c_{max} will not be reachable by the greedy search.

4.2 Multi-Agent Graph RRT*

The multi-agent version of G-RRT* (MA-RRT*) allows us to solve cooperative pathfinding problems where agents move on motion graphs. The formulation of the algorithm is identical to the G-RRT*, execept that the algorithm performs the search in the agents' joint-state space. This leads to the following modifications. A state in the MA-RRT*'s state space becomes n-tuple$(w_1, \ldots, w_n)$, where w_i is the waypoint occupied by the agent i. The starting state x_{init} is an n-tuple $\mathbf{s} = (s_1, \ldots, s_n)$, where s_i is the start waypoint of agent i. Similarly, the target region contains only one state $\mathbf{d} = (d_1, \ldots, d_n)$, where d_i is the target waypoint of agent i. The SAMPLE procedure returns an n-tuple $(w_1, \ldots, w_n)$, where w_i is a waypoint for agent i taken at random from the agent's motion graph G_i^M. The EXTEND procedure is identical to that of G-RRT*, but instead of states representing position of one agent, here they represent the position of all agents. The greedy search is performed by generating a sequence of joint actions as described in Algorithm 4. After each joint-action is generated in the greedy search, the n-tuple of action is checked for possible separation breach between the agents. This ensures that the paths of individual agents stay separated. Typical outcome of such a greedy search is a path that leads each of the agents to his target waypoint and when the target is reached, the agents wait for the longest traveling agent. Note that we do not specify the time at which such a state should be achieved.

For a good function of the RRT* algorithm, one needs to specify a suitable distance function to influence the behavior of NEAR and NEAREST functions. Here we define the distance between two joint-states $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ as $\sum_{i=1}^{n} cost_i^{LB}(x_i, y_i)$, where $cost_i^{LB}(x, y)$ is the lower bound on the cost of move from x to y for agent i. Since we define cost in terms of travel time, a suitable definition of such a function is $\frac{dist(x,y)}{maxspeed_i}$, where $dist(x, y)$ is the distance between waypoints x and y and $maxspeed_i$ is the maximal speed at which agent i can move. We chose such a function, because it represents a lower-bound on the cost of transition between two joint-states. To see why, imagine two agents moving from the joint state (s_1, s_2) to joint state(d_1, d_2). If $dist(s_1, d_1) < dist(s_2, d_2)$ then the first agent will arrive to d_1 before the second agent arrives tod_2 and the first agent will have to wait for the second agent. Now, recall that the solution quality metric we use is the sum of time the agents spent outside their goal positions. Therefore, waiting for other agents contributes to the global solution cost, with single exception when the agent is on its target waypoint, where the waiting is "free". Therefore, the sum of minimum travel times for each agent represents a correct lower bound on the cost of transition between two joint states with respect to our solution quality metric.

Lemma 2. *The MA-RRT* algorithm inherits convergence, soundness, completeness and optimality properties from the G-RRT* algorithm.*

Algorithm 4 GREEDY($G_M, \mathbf{s}, \mathbf{d}$)

```
1: x ← s; c ← 0; path ← (∅, ..., ∅)
2: while x ≠ d and c ≤ c_max do
3:     (path_i, ..., path_n) ← path
4:     for all x_i ∈ x do
5:         N ← children(G_M, x_i)
6:         x' ← arg min_{x ∈ children(G_M, x_i)} h(x_i)
7:         c ← c + cost(x_i, x_i'); path_i ← path_i ∪ (x_i, x_i');
8:         x_i ← x_i'
9:     end for
10:    if not COLLISIONFREE(path_1, ..., path_n) then
11:        return path
12:    else
13:        path ← (path_1, ..., path_n)
14:    end if
15: end while
16: return (x, path)
```

Proof. Follows from the fact, that the distance in the multi-agent case is a proper metric and satisfies the triangle inequality, otherwise the Cartesian product of the individual motion graphs can be converted into a new graph with joint vertices, that is to the joint-state space graph. □

4.3 Informed Sampling

The above presented version of MA-RRT* samples the agents' joint-state space with uniform probability distribution. The performance of the algorithm (on average problem instance) can be improved by instructing the algorithm that some regions of the state space are more likely to contain high-quality solutions than others. This can be done by changing the distribution of samples so that the algorithm samples more frequently the promising regions of the state space.

For sparse instances of cooperative pathfinding problems, the global solutions typically consist of paths that are identical to the agent's optimal path in the absence of other agents or contain only minor diversions from the agent's optimal path. Therefore, we propose a sampling strategy that strongly favors regions around optimal paths of the individual agents. We call this informed-sampling version of MA-RRT (isMA-RRT).

The isMA-RRT algorithm runs side by side a number of G-RRT* solvers for each agent for finding their locally optimal paths and one MA-RRT* planner for finding a global solution to the multi-agent problem. The pseudocode of isMA-RRT solver is in Algorithm 5. First, the single-agent planners are iterated in order to find a path for each agent involved in the problem. When the single-agent paths are known for each agent, we start iterating the multi-agent solver with a modified sampling function, which takes the samples from the Gaussian neighborhood of the single-agent paths as show in Algorithm 6.

Algorithm 5 isMA-RRT*

while not interrupted **do**
  for i =1 . . . n **do**
    run one iteration of G-RRT* solver for agent i
  end for
  if all single-agent solver found some path **then**
    run iteration of MA-RRT* with modified sampling
  end if
end while

Algorithm 6 SAMPLE($G^M, (path_1, \ldots, path_n), \sigma$)

$t_{max} \leftarrow$time when the last agent arrives to its destination
$t \leftarrow$uniform random from $(0, t_{max})$
for $i = 1 \ldots n$ **do**
  $(x, y) \leftarrow path_i(t)$
  $x \leftarrow x + N(0, \sigma)$; $y \leftarrow y + N(0, \sigma)$
  $w_i \leftarrow$nearest vertex in G^M to position (x, y)
end for
return $(w_1, \ldots, w_n)$

4.4 Other Optimization

Practical implementations of sampling based algorithms typically contain two other optimizations. Firstly, it is often beneficial to sample the target state with a given probability, since it promotes the growth of the tree in the direction of the goal. Secondly, once a new solution is found one can use it as an upper bound for the subsequent sampling. This means that a new vertex v will not be added to the tree if the cost of the vertex together with the lower-bound estimate on the path from v to the goal region exceeds the cost of the currently best solution. In our implementation, we use both these techniques.

5 Evaluation

We compared the performance of the unbiased version of MA-RRT* (MA-RRT*) and informed-sampling MA-RRT* (isMA-RRT*) with A* search in joint-state space (JA) and optimal anytime algorithm (OA) in terms of scalability and solution quality. All three algorithms were implemented in Java in a one common framework. Where possible, the implementations of the individual algorithms shared code. I.e. all algorithms use identical collision-checking routine, graph structures etc.

We evaluated the performance of the algorithms on the following set of synthetic problem instances. The agents move on a square-shaped grid-like motion graph, where the waypoints are placed on the grid having the step of 1 meter and the motion primitives are straight moves at the constant speed of 1 m/s connecting the vertices in 4-neighborhood. Further, a single second long waiting motion primitive is available at each waypoint. Random ten percent of the vertices of the motion graph is removed to

represent obstacles. A unique start waypoint and unique destination waypoint is chosen randomly for each agent. Further, for each such instance we check if all agents can reach their destination to ensure that the instance admits a solution.

Obviously, this represents a commonly used unit-grid modeled in the framework of motion graphs. The main difference is that in unit grids the agents "jump" between the vertices, here they follow a trajectory defined by the motion primitive and thus the collision checking is done on the actual trajectories of the agents. We chose such a representation for two reasons. First, it allows us to set the separation distance independently to the size of grid as is common in many domains, e.g. in UAV conflict resolution. Second, since the moves between the points in the grid are modeled as proper motion primitives, we expect that despite the simplicity of our experimental environment the results will generalize better to more complex real-world domains.

Our set of problem instances contained instances that varied in the size of the grid and in the number of agents. We used the following values of the two parameters. Grid sizes: 10x10, 30x30, 50x50, 70x70, 90x90. Numbers of agents: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. The separation distance was set to a constant 0.8[4]. The problem instance set contains 120 random instances (with random obstacles and random start and destination positions) for each combination of the grid size and the number of agents. Thus, in total the experiment contained 6000 different problem instances. Each of the algorithms was executed on every instance with the runtime limit of 5 seconds. An illustrative example of one such instance is in Figure 1. The experiments were executed in HotSpot 1.6 64-bit Java VM running on AMD FX-8150 3.6 GHz CPU.

5.1 Results

To convey how successful were the algorithms on our problem instance set, we plot the performances curves (proposed in [12]) for each algorithm. We record the runtime to find the first valid solution to the problem instance for each algorithm. Then, we sort the instances according to the that runtime for each algorithm independently. The results are plotted in Figure 2. On the x-axis is the index of instance in the algorithm's sorted sequence, on the y-axis is the runtime the algorithm needed to find the first solution to that problem instance. It should be noted that the ordering of the instances is different for each algorithm. The x-position of the last point in the performance curve can be interpreted as the number of instances of the total 6000 instances the algorithm solved in the runtime limit of 5 seconds. We can see that JA resolved 21% of the instances, OA 38%, MA-RRT* 56% and bsMA-RRT* 77 % of instances from our problem instance set.

Figure 3 show the comparison of relative solution quality for the anytime algorithms, JA is not plotted since it always returns optimal solutions. For all algorithms we show the quality of the first returned solution and the quality of the best solution found in the 5 second runtime limit. The suboptimality is measured only on a subset of instances for which either JA or OA returned provably optimal solution, which was in our case 2348 instances. The suboptimality measure is expressed in percent as

[4] We also tried to vary separation distance and the average size of an obstacles, but found that these parameters have no significant influence on the scalability of the algorithms.

$$\text{suboptimality} = \left(\frac{\text{cost of returned solution}}{\text{cost of optimal solution}} - 1 \right) \cdot 100.$$

Figure 4 (left) shows the percentage of successfully solved instances for different widths of the grid. The dependence is shown in sections for three different numbers of agents. Similarly, Figure 4 (right) shows the percentage of successfully solved instances for increasing number of agents. Again, we show the values for three different grid sizes. Each of the plotted points represents an aggregation from 120 random instances.

The plots indicate how the individual algorithms scale with respect to the size of environment and the number of agents. As expected, the A* search in joint-state space scales poorly with the number of agents. In fact, we observed that there are two typical cases when JA succeeds to find a solution in the limited time. Firstly, JA is able to solve instances with two or three agents relatively easily, since these are relatively low exponents. See Figure 4 (right) for the clear drop of success rate for the numbers of agents above 3. Secondly, JA seems to be successful in instances that do not contain local minima and thus can be solved simply by following the heuristics. An example of such an instance is when the agents have no obstacles to avoid on the paths to their destinations and further their are paths are mutually conflict-free. We can see that even in large environments (3rd plot on the right in Figure 4), there are some instances that JA was able to solve. However, for all other cases JA fails to scale.

An interesting case is the scalability of the OA algorithm. The main idea behind the algorithm is to avoid expensive planning in the joint-state space by performing a number of single-agent searches for alternative paths in presence of dynamic obstacles (other agents). Our experiments show that the idea works nicely for the small environments, but it starts to collapse in larger environments, since in large environments each of such searches takes significant time that may be higher than focused joint-state space search (see Figure 4, left).

The sampling-based algorithms MA-RRT* and isMA-RRT*, on the other hand, show no such performance decline when the size of the environment grows. On the contrary, we can see that in the scenarios with many agents, the performance of the algorithms drops if the space is too "tight". That is, the algorithm works best if the space is sufficiently large for the number of agents.

6 Conclusion

In this paper we proposed MA-RRT*, a novel anytime algorithm for solving cooperative pathfinding problems. Unlike other state-of-the art approaches, which are build upon forward-search, our technique exploits the new advancements in sampling-based motion planning and searches for a solution in agents' joint-state space using randomized sampling. We compared the performance of our approach with the A* search in joint-state space and with the current state-of-the-art anytime algorithm. Our experiments demonstrate the limits of the forward-search based approaches to cooperative pathfinding in large, but sparse environments. Our results show that these instances can be efficiently solved using one of our sampling-based algorithms for the price of a slight decrease in the solution quality.

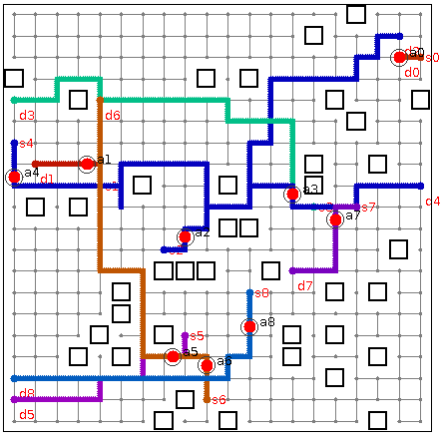

Fig. 1. An example problem instance

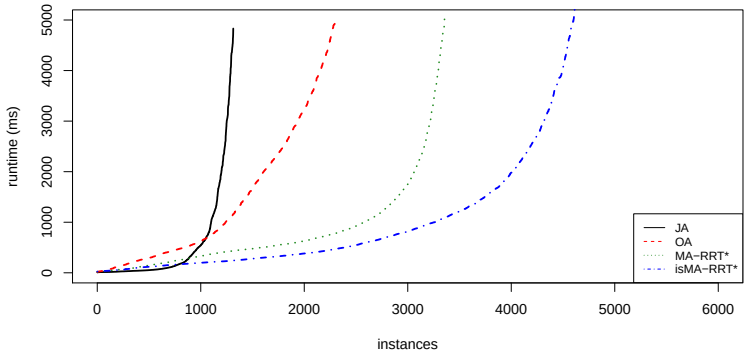

Fig. 2. First-solution performance curve

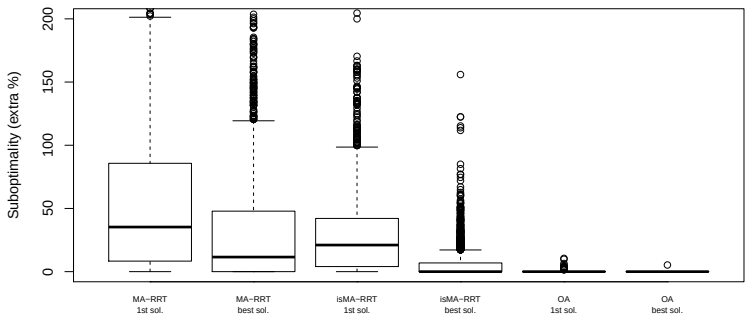

Fig. 3. Solution quality

As a by product, we translated the original RRT* algorithm for continuous state spaces to graphs and devised a new algorithm for finding shortest path on spatial graphs using randomized sampling.

The presented results admit various direction of future work. Our algorithm represents an efficient anytime method for search in joint-state spaces. It can be seen as a scalable, sampling-based counterpart of the JA algorithm. We are currently working on a sampling-based counterpart of the OA algorithm, i.e. on an extension of MA-RRT* that will change its sampling strategy based on the detected independences.

In this paper, we have formulated the algorithm on a motion graph, but the approach admits a straightforward extension to continuous state spaces. This can be done by sampling the continuous joint-state space of all agents and by using straight-line visibility checks in the place of greedy search procedure.

Acknowledgements

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS13/143/OHK3/2T/13 and by the Ministry of Education, Youth and Sports of Czech Republic within the grant no. LD12044. Further, we would like to thank Michal Štolba for his help with the experimental evaluation of the presented approach.

References

1. Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. Motion planning in urban environments: Part ii. In *IROS*, pages 1070–1076, 2008.
2. David Ferguson and Anthony (Tony) Stentz. Anytime rrts. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pages 5369 – 5375, October 2006.

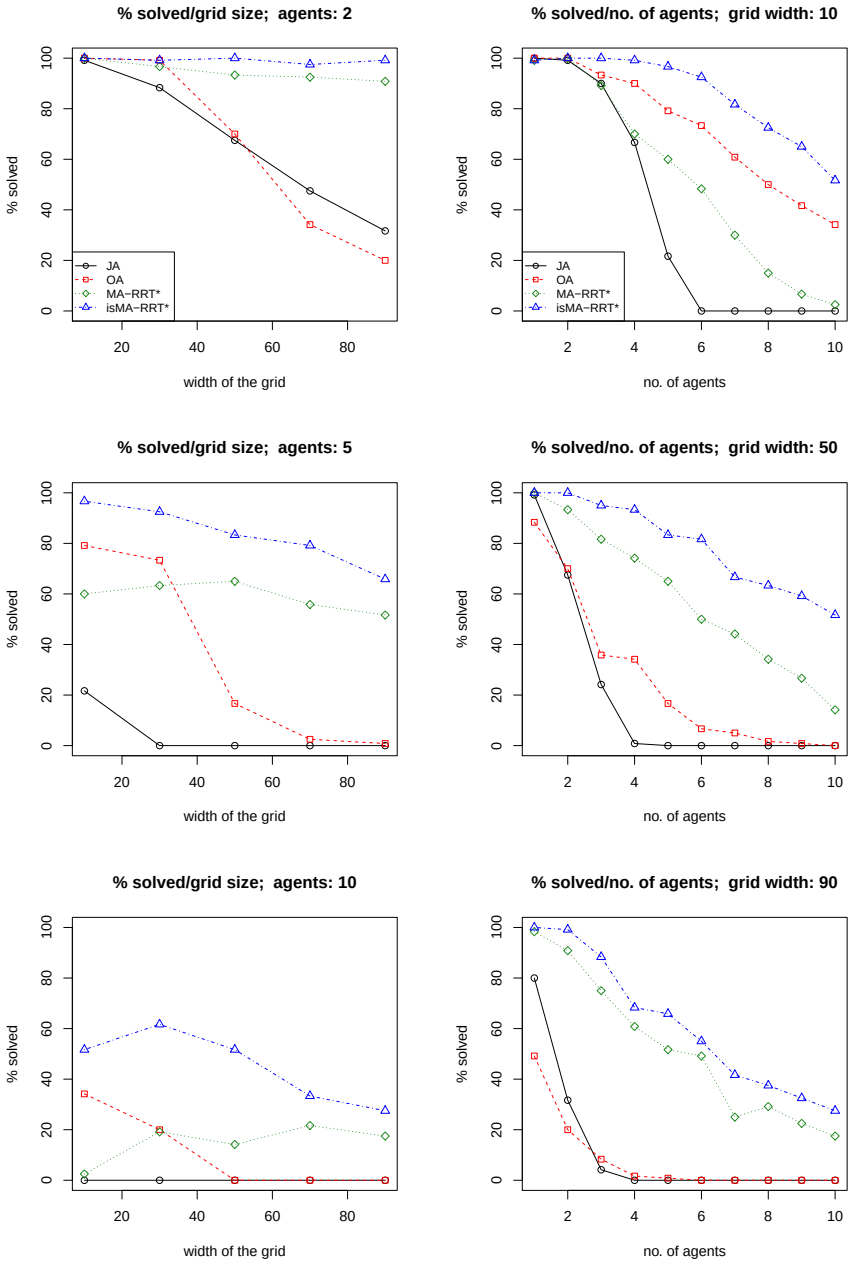

Fig. 4. Scalability in the size of environment (left) and in the number of agents (right)

3. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.
4. J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, December 1984.
5. Myung Hwangbo, James Kuffner, and Takeo Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. In *ICRA*, pages 1035–1041, 2007.
6. Shotaro Kamio and Hitoshi Iba. Random sampling algorithm for multi-agent cooperation planning. In *IROS*, pages 1265–1270, 2005.
7. Karaman and Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, June 2011.
8. Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *I. J. Robotic Res.*, 20(5):378–400, 2001.
9. Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In *IJCAI*, pages 294–300, 2011.
10. Michael Otte and Nikolaus Correll. Any-com multi-robot path-planning with dynamic teams: Multi-robot coordination under communication constraints. In *Proc. 12th International Symposium on Experimental Robotics*, 2010.
11. Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.
12. Trevor Scott Standley. Finding optimal solutions to cooperative pathfinding problems. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.
13. Trevor Scott Standley and Richard E. Korf. Complete algorithms for cooperative pathfinding problems. In Toby Walsh, editor, *IJCAI*, pages 668–673. IJCAI/AAAI, 2011.
14. Pavel Surynek. A novel approach to path planning for multiple robots in bi-connected graphs. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 928–934, Piscataway, NJ, USA, 2009. IEEE Press.