Sparse Real-time Decision Diagrams for Continuous Multi-Robot Path Planning

Pavel Surynek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czechia
pavel.surynek@fit.cvut.cz, 0000-0001-7200-0542

***Abstract*—Multi-robot path planning (MRPP) is the task of finding non-conflicting paths for robots via which they can navigate themselves to specified individual goal positions. MRPP uses an undirected graph to represent a shared environment in which the robots move instantaneously between vertices in discrete time steps. Such discrete formulation enables relatively simple algorithms, often based on multi-valued decision diagrams (MDDs) that represent possible paths for each robot, but results in an inaccurate modeling of the real robotic task. Recently introduced continuous variant of MRPP assumes fixed trajectories for robots and fully continuous time but is more difficult to be addressed algorithmically. The set of possible paths for individual robots in the continuous variant can be represented in real-time decision diagram (RDD) which however is often too large. An improvement of RDDs based on sparsification that includes paths into RDD according to their heuristic prioritization is suggested in this short paper. We show that sparse RDDs can improve existing compilation-based algorithms significantly while keeping their optimality guarantees.**

***Index Terms*—multi-agent path planning, continuous time, continuous space, decision diagrams, real-time, sparsification, sum-of-costs optimality**

Multi-robot path planning in graphs (MRPP) represents fundamental problem in combinatorial motion planning in robotics [1]–[5]. The task is to navigate robots from their initial positions to specified individual goal positions. The environment is usually modeled as an undirected graph where vertices represent positions and robots move between vertices across edges. The two requirements make the problem challenging: **(1)** the robots must not collide with each other, that is they neither can share a vertex nor they can traverse an edge in opposite directions and **(2)** some objective such as the total number of actions must be optimized.

We address the MRPP problem from the perspective of knowledge compilation based on decision diagrams. Across variety of algorithms for MRPP it is needed to represent a large set of candidate paths for each robot out of which a specific algorithm selects a non-conflicting collection, one path for each robot. This representation has been successfully applied for the discrete variant of MRPP via *multi-valued decision diagrams* (MDDs) in both search-based [6], [7] and compilation-based algorithms [8].

In the continuous variant of MRPP [9], [10], where robots move smoothly along fixed continuous trajectories, an analogous concept to MDDs appeared as *real-time decision diagrams* (RDDs) only recently, however it is less efficient in terms of compression than MDDs [11]. The efficiency of compression in MDDs for the discrete variant of MRPP is rooted in the property that a single node in MDD is often shared by many represented paths since the corresponding vertex can be visited at discrete times only. This effect is however less frequent when continuous time is used since a vertex can be visited at many different continuous times which results in many nodes in RDD.

Therefore we suggest in this paper a sparsification technique for RDDs that instead of representing all possible candidate paths represents only a relevant subset of them. The important feature of the technique is that optimality guarantees of MRPP solving algorithms build on top of sparse RDDs are kept.

The paper is organized as follows: first, a continuous variant of MRPP is introduced. Then, a real-time decision diagrams (RDD) for MRPP are recalled followed by the new sparsification technique. A compilation-based algorithm that compiles MRPP to propositional satisfiability (SAT) and uses nodes from RDDs for construction of Boolean decision variables is introduced as next. Finally, an experimental evaluation that compares SAT-based algorithm using sparse and full RDDs and search-based conflict-based search is presented.

I. Multi-robot Path Planning with Continuous Space and Time

We follow the definition of MRPP with continuous space and time denoted MRPP$^{\mathcal{R}}$ from [10]. MRPP$^{\mathcal{R}}$ shares components with the discrete MRPP: undirected graph $G = (V, E)$, set of robots $R = \{r_1, r_2, ..., r_k\}$, and the initial and goal configuration of robots: $s_0 : R \rightarrow V$ and $s_+ : R \rightarrow V$, that assign robots vertices of the graph. Various continuous extensions of MRPP are possible. However the property that is important for the following concepts and algorithms is that vertices V are assigned fixed positions in a continuous space and edges E are assigned fixed continuous trajectories interconnecting the positions. It is also needed to know how the robot traverses the trajectory in time to be able to calculate collisions between robots' bodies.

In the rest of the text we will use a simple 2D variant of MRPP$^{\mathcal{R}}$ consisting of circular robots with constant speed and instant acceleration. However it is important to note that all developed concepts can be generalized for variants of MRPP$^{\mathcal{R}}$

taking place in more complex space and using more complex kino-dynamic properties of robots.

Definition 1. *(**MRPP**$^{\mathcal{R}}$) Multi-robot path finding with continuous time and space is a 5-tuple* $\Sigma^{\mathcal{R}} = (G = (V, E), R, s_0, s_+, \rho)$ *where* G, R, s_0, s_+ *are from the discrete MRPP and* ρ *determines continuous extensions:*

- $\rho.x(v), \rho.y(v)$ *represent the position of vertex* $v \in V$ *in the 2D plane*
- $\rho.speed(r)$ *determines constant speed of robot* $r \in R$
- $\rho.radius(r)$ *determines the radius of robot* $r \in R$*; we assume that robots are circular discs with omni-directional ability of movements*

The major difference from the standard discrete MRPP where robots move instantly between vertices (disappears in the source and appears in the target instantly) is that smooth continuous movement between a pair of vertices (positions) along the given trajectory (a straight line in the case of 2D variant) interconnecting the vertices takes place in MRPP$^{\mathcal{R}}$. Hence we need to be aware of the presence of robots at some point in the space (2D plane) at any time.

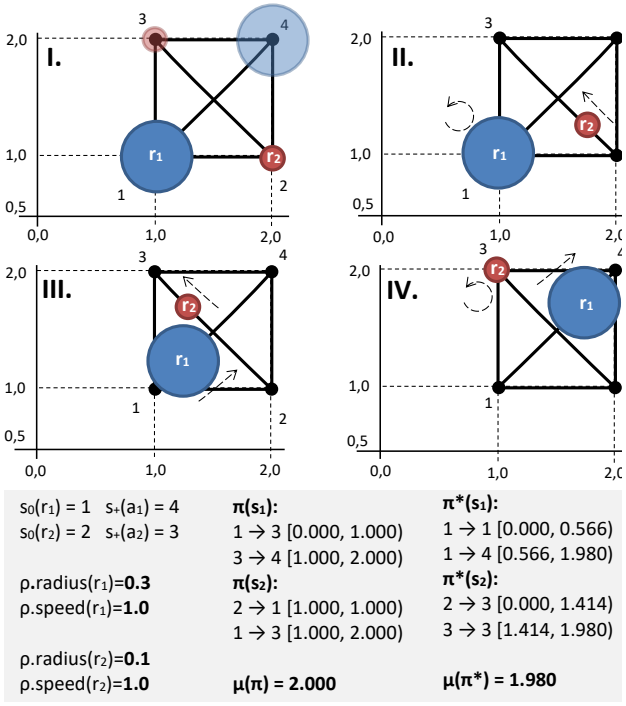

Fig. 1. An example of MRPP$^{\mathcal{R}}$ instance with two robots. A feasible makespan/sum-of-costs sub-optimal solution π (makespan $\mu(\pi) = 2.0$) and makespan/sum-of-costs optimal solution $\pi*$ (makespan $\mu(\pi*) = 1.980$) are shown.

Collisions may occur between robots in MRPP$^{\mathcal{R}}$ due to their volume; that is, they collide whenever their bodies **overlap**. In contrast to MRPP, collisions in MRPP$^{\mathcal{R}}$ may occur not only in a vertex or edge being shared by colliding robots but also on pairs of vertices or edges (lines interconnecting positions assigned to vertices) that are too close to each other and simultaneously traversed by large robots.

We can further extend the continuous properties by introducing the direction of robots and the need to rotate robots towards the target vertex before they start to move. Also robots can be of various shapes not only circular discs [12] and even can change their shape in time.

A solution to given MRPP$^{\mathcal{R}}$ $\Sigma^{\mathcal{R}}$ is a collection of temporal plans for individual robots $\pi = [\pi(r_1), \pi(r_2), ..., \pi(r_k)]$ that are **mutually collision-free**. A temporal plan for robot $r \in R$ is a sequence $\pi(r) = [((s_0(r), s_1(r)), [t_0(r), t_1(r))); ((s_1(r), s_2(r)), [t_1(r), t_2(r))); ...; ((s_{m(r)-1}(r), s_{m(r)}(r)), [t_{m(r)-1}(r), t_{m(r)}(r))))]$ where $m(r)$ is the length of individual temporal plan and $t_{m(r)}(r)$ is its duration. Each pair $(s_i(r), s_{i+1}(r)), [t_i(r), t_{i+1}(r)))$ corresponds to traversal event between a pair of vertices $s_i(r)$ and $s_{i+1}(r)$ starting at time $t_i(r)$ and finished at time $t_{i+1}(r)$.

It holds that $t_i(r) < t_{i+1}(r)$ for $i = 0, 1, ..., m(r) - 1$. Moreover consecutive events in the individual temporal plan must correspond to edge traversals or waiting actions, that is: $\{s_i(r), s_{i+1}(r)\} \in E$ or $s_i(r) = s_{i+1}(r)$; and times must reflect the speed of robots for non-wait actions.

The duration of individual temporal plan $\pi(r)$ is called an *individual makespan*; denoted $\mu(\pi(r)) = t_{m(r)}(r)$. The overall *makespan* of π is defined as $\max_{i=1}^{k}\{\mu(\pi(r_i))\}$. The individual makespan is sometimes called an *individual cost*. A *sum-of-cost* for plans π is defined as $\sum_{i=1}^{k} \mu(\pi(r_i))$.

An example of MRPP$^{\mathcal{R}}$ and makespan/sum-of-costs optimal solution is shown in Figure 1.

Via reduction of MRPP to MRPP$^{\mathcal{R}}$ it can be observed that finding the makespan as well as the sum-of-costs optimal solution with continuous time is an NP-hard problem [13], [14].

II. RDDs: Real-time Decision Diagrams

We further elaborate the concept of *real-time decision diagram* (RDD) in this paper [11]. RDD is analogous to *multi-valued decision diagram* (MDD) [6], [7] that in the context of MRPP is used to represent a set of discrete paths in a compact way.

MDD is a layered structure where the l-th layer consists of copies of some vertices from V called nodes, denoted (u, l) for $u \in V$. Directed edges are introduced to connect nodes from consecutive layers in MDD and must correspond to edges in E, that is, whenever nodes (u, l) and $(v, l+1)$ are connected in MDD, there must be a corresponding edge $\{u, v\} \in E$. In addition to this, there are directed edges in MDD that interconnects nodes corresponding to identical vertices from V, that is, edges from (v, l) and $(v, l+1)$ for any $v \in V$ and any relevant l. The construction of MDD ensures that any directed path in MDD starting at the first layer and finishing at the last layer corresponds to a movement of a robot in G (a single vertex may be visited multiple times or the robot may wait in a vertex, which is enabled by the second type of edges). We will refer to the sequence of vertices visited by the robot as a path.

The advantage of MDDs for MRPP is that even if a vertex v is reachable via multiple paths at a given time

step l it is represented only once using single node (v, l) in MDD. This simple property enables significant compression of representation of the set of paths. Hence, we usually need only a small MDD to represent all paths interconnecting starting position of a robot with its goal position.

In MRPP solving algorithms, MDD is assigned to each robot r_i, denoted MDD_i, that represents a set of possible paths for r_i. In a very simplistic sense, algorithms then try to select paths from individual MDDs of robots so that these paths are pairwise non-conflicting resulting in a valid joint plan for all robots from R.

Similarly RDD_i defines for a robot r_i its possible space-time positions and movements. However, it no longer makes sense to speak about layers in RDDs since time is no longer discretized.

Formally, RDD_i is a directed graph (X^i, E^i) where X_i consists of pairs (u, t) with $u \in V$ and $t \in \mathbb{R}_0^+$ is time and E_i consists of directed edges of the form $((u, t_u); (v, t_v))$. An edge $((u, t_u); (v, t_v))$ corresponds to robot's movement from u to v started at time t_u and finished at time t_v (times t_u and t_v must be consistent with r_i's speed). Similarly as in MDD, a directed path from a node (u, t) to a node (v, t') in RDD_i should correspond to a trajectory of r_i in G embedded in 2D plane.

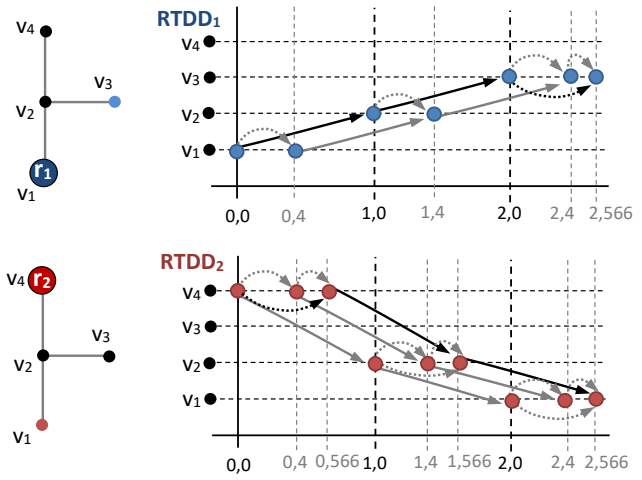

Fig. 2. An example of real-time decision diagrams for $\text{MRPP}^{\mathcal{R}}$ with two robots.

Similarly as in MDD, waiting in $u \in V$ can be expressed by introducing edge $((u, t_u); (u, t'_u))$ ($t'_u - t_u$ is the duration of wait action). A pair $(s_0(r_i), 0) \in X_i$ indicates the start and $(s_+(r_i), t)$ for some t corresponds to reaching the goal position at time t.

RDDs for individual robots can be constructed with respect to collision avoidance constraints. If there is no collision avoidance constraint then RDD_i simply corresponds to a set of shortest temporal plans for robot r_i. But if a collision avoidance constraint is present, say $(r_i, (u, v), [\tau_0, \tau_+))$, whose interpretation is that r_i cannot start traversing edge (u, v) at any time $t \in [\tau_0, \tau_+)$, we need to allow robot r_i to wait in u until the unsafe interval $[\tau_0, \tau_+)$ passes. It terms of RDDs, waiting in u until τ_+ corresponds to adding a node (u, τ^+) and edge $((u, \tau^0); (u, \tau^+))$ to RDD_i.

Similarly for wait constraints $(r_i, (u, u), [\tau_0, \tau_+))$ that forbid waiting in u during $[\tau_0, \tau_+)$. In such a case, we need to anticipate the constraint before entering u, that is we can wait until $\tau_+ - t_x$ in the source vertex before entering u where t_x is the time needed to traverse the edge towards u.

The process of building RDDs is described in details in [11]. A detailed reasoning about unsafe intervals and how paths can be constructed using wait actions to avoid them is described in details in [15]. An example of RDDs is shown in Figure 2.

A. Sparsification in RDDs

RDDs can be used to represent candidate paths for individual robots within the *conflict-based search algorithm* (CBS) [16], or to be more precise in CCBS [10], a continuous variant of CBS. Initially, each robot has a set of candidate paths with respect to empty set of collision avoidance constraints (there paths are shortest path connecting robots' initial and goal positions). The algorithm selects a path for each robot and then validates the selected paths with respect to collisions. If there is no collision, then the algorithm returns a valid optimal MRPP solution. However if a collision is detected, say between robots r_i and r_j, then collision avoidance constraints are added for colliding robots, calculated using their geometric and kinematic properties described by ρ - for each robot, the minimum waiting time to avoid the collision with other robot is determined and stored as an unsafe interval associated with the robot and edge traversal (or wait action). A *mutex* is stored so that the pair of conflicting movements never occur simultaneously again. The knowledge of the detected collision is stored in RDDs and path selection from RDDs is repeated.

As the algorithm proceeds, the set of collision avoidance constraints for individual robots grows and to keep soundness of the solving process each RDD must hold all paths with respect to **any subset** of collision avoidance constraints since a-priori it is not known what set of collisions a given robot will avoid. This could lead to significant growth in the size of RDDs and as a consequence the high-level path selection must search large space of candidate paths defined by RDDs.

Therefore we introduce a **sparsification** technique for RDDs. The sparsification has been recently introduced for MDDs [17]. The core idea of sparsification is not to represent all candidate paths in RDDs with respect to a given set of collision avoidance constraints but only a subset of them. The subset of paths can be chosen to prefer paths that are more likely to be conflict-free. The choice of paths to be represented in the sparse RDD is a room for integrating MRPP-specific heuristics. This is especially important for compilation-based algorithms for solving MRPP as the internal decisions of the target solver are difficult to influence.

Similar to RDDs called *pool of paths* has been used in the context of mixed integer linear programming (MILP)-based compilation for MRPP [18], [19] which however does not explicitly focus on sparsification.

B. Candidate Path Selection Heuristic

Preliminary experiments indicate that increasing the number of conflict avoidance constraints that are taken into account during construction of RDDs for individual robots is a promising strategy since often a valid solution distributes collision avoidances between robots equally while it is not much frequent that single robot avoids many collisions.

In other words, RDDs are constructed first ignoring collision avoidance constraints (zero collision avoidance constraints per path are considered), then 1 collision avoidance constraint per path is considered, and so on eventually finishing with the full RDD in which paths with respect to any subset of collision avoidance constraints are represented.

Sparse RDDs are shown in Figure 3. The right part shows RDDs after extension via considering new conflict aviodance constraints.

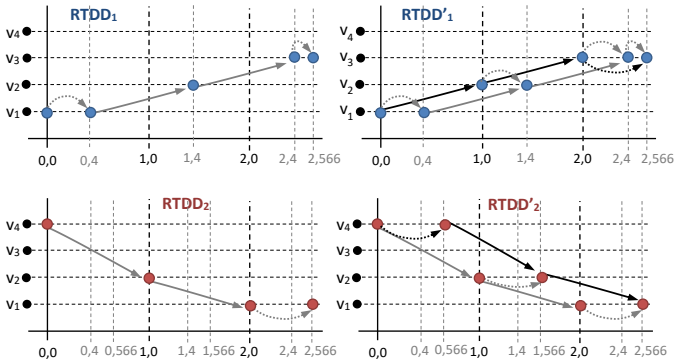

Fig. 3. An example of sparse real-time decision diagrams for MRPP$^{\mathcal{R}}$ from Figure 3.

III. Compilation via RDDs

Representation of paths via RDDs and their sparse variant has been integrated in the SMT-CBS framework [20]. SMT-CBS uses a SAT solver [21] for selecting a collection of non-conflicting paths from candidate paths represented in individual RDDs. During this, mutexes that model conflicting pairs of movements are taken into account.

To encode the selection of paths as a SAT problem, we introduce a decision Boolean variable for each node and edge in RDDs. Having RDDs $[RDD_1, ..., RDD_k]$; $\text{RDD}_i = (X^i, E^i)$: we have a variable $\mathcal{X}_u^t(r_i)$ for each $(u,t) \in X^i$ and $\mathcal{E}_{u,v}^{t_u,t_v}(r_i)$ for each directed edge $((u,t_u),(v,t_v)) \in E^i$. The meaning of the variables is that $\mathcal{X}_u^t(r_i)$ is $TRUE$ if and only if the robot r_i appears in u at time t and similarly for edges: $\mathcal{E}_{u,v}^{t_u,t_v}(r_i)$ is $TRUE$ if and only if r_i moves from u to v starting at time t_u and finishing at t_v.

MRPP$^{\mathcal{R}}$ rules are encoded on top of these variables so that eventually we want to obtain a formula $\mathcal{F}(\mu)$ that encodes existence of a solution of makespan μ to given MRPP$^{\mathcal{R}}$. We need to encode that robots do not skip but move along edges, do not disappear or appear from nowhere etc. For details, how $\mathcal{F}(\mu)$ is constructed we refer the reader to [11].

We a-priori do not add constraints for eliminating collisions; these are added **lazily** after assignment/solution validation.

Algorithm 1: High-level of SMT-CBS$^{\mathcal{R}}$ for the sum-of-costs objective.

```
1  SMT-CBS^R (Σ^R = (G = (V, E), A, s_0, s_+, ρ))
2      constraints ← ∅
3      π ← {π*(r_i) a shortest temporal plan from s_0(r_i) to
         s_+(r_i) | i = 1, 2, ..., k}
4      μ ← max_{i=1}^{k} μ(π(r_i))
5      ξ ← Σ_{i=1}^{k} μ(π(r_i))
6      while TRUE do
7          (π, constraints, μ_next, ξ_next) ←
             SMT-CBS-Fixed^R(Σ^R, constraints, μ, ξ)
8          if π ≠ UNSAT then
9              return π
10         μ ← μ_next
11         ξ ← ξ_next
```

Hence, $\mathcal{F}(\mu)$ constitutes an *incomplete model* for the input MRPP instance: the MRPP instance is solvable within makespan μ then $\mathcal{F}(\mu)$ is satisfiable. The opposite implication however does not hold since a MRPP solution corresponding to the satisfying assignment of $\mathcal{F}(\mu)$ may lead to a collision between robots.

A. Optimal Algorithm Based on Sparse RDDs

The SMT-CBS algorithm itself is divided into two procedures: SMT-CBS$^{\mathcal{R}}$ representing the main loop (Algorithm 1) and SMT-CBS-Fixed$^{\mathcal{R}}$ solving the input MRPP$^{\mathcal{R}}$ for a fixed maximum makespan μ and sum-of-costs ξ.

The main loop iteratively increases μ and ξ following the style of SATPlan [22] - trying plans of increasing length until sufficient length is reached. The SMT-CBS algorithm relies on the fact that the solvability of MRPP$^{\mathcal{R}}$ w.r.t. cumulative objective like the sum-of-costs or makespan behaves as a non decreasing function. Hence trying increasing makespan and sum-of-costs eventually leads to finding the optimum provided we do not skip any relevant value [11].

A formula $\mathcal{F}(\mu)$ according to the given sparse RDDs with respect to collected collision avoidance constraints is constructed in SMT-CBS-Fixed$^{\mathcal{R}}$. New collisions are resolved **lazily** by adding *mutexes* (disjunctive constraints). A collision is avoided in the same way as in CCBS; that is, one of the colliding robot waits. Collision eliminations are tried until a valid solution is obtained or until a failure for current RDDs is encountered.

In the case of a failure, an attempt to extend RDDs is done by allowing for considering more conflict avoidance constraints for individual robots (or by using any other heuristics). If RDDs are successfully extended, the process is repeated with these new RDDs. Otherwise, that is when we already have the full RDDs for current μ, it means there is no solution for current μ and we need to try greater μ in the next iteration of the main loop (lines 6-11 of SMT-CBS$^{\mathcal{R}}$).

For resolving a collision we need to: **(1)** eliminate simultaneous execution of colliding movements and **(2)** augment the formula to enable avoidance (waiting). Assume a collision between robots r_i traversing (u_i, v_i) during $[t_i^0, t_i^+)$

and r_j traversing (u_j, v_j) during $[t_j^0, t_j^+)$ which corresponds to variables $\mathcal{E}_{u_i,v_i}^{t_i^0,t_i^+}(r_i)$ and $\mathcal{E}_{u_j,v_j}^{t_j^0,t_j^+}(r_j)$. The collision can be eliminated by adding the following **mutex** (disjunction) to the formula: $\neg\mathcal{E}_{u_i,v_i}^{t_i^0,t_i^+}(r_i) \vee \neg\mathcal{E}_{u_j,v_j}^{t_j^0,t_j^+}(r_j)$ (line 22 of SMT-CBS-Fixed$^{\mathcal{R}}$). Satisfying assignments of the next $\mathcal{F}(\mu)$ can no longer lead to this collision. Next, the formula is augmented according to new RDDs that reflect the collision - decision variables and respective constraints are added.

After resolving all collisions we check whether the sum-of-costs bound is satisfied by plan π. This can be done easily by checking if $\mathcal{X}_u^{t_u}(r_i)$ variables across all robots together yield higher cost than ξ or not. If cost bound ξ is exceeded then corresponding nogood is recorded and added to $\mathcal{F}(\mu)$ (lines 11-14 of SMT-CBS-Fixed$^{\mathcal{R}}$) and the algorithm continues by searching for a new satisfying assignment to $\mathcal{F}(\mu)$. The nogood says that $\mathcal{X}_u^{t_u}(r_i)$ variables that jointly exceed ξ cannot be simultaneously set to $TRUE$.

The set of pairs of collision avoidance constraints is propagated across entire execution of the algorithm. Constraints originating from a single collision are grouped in pairs so that it is possible to introduce mutexes for colliding movements discovered in previous steps.

Algorithm 2: Low-level of SMT-CBS$^{\mathcal{R}}$

```
1  SMT-CBS-Fixed^R(Σ^R, cons, μ, ξ)
2      RDD ← build-RDDs(Σ^R, cons, μ)
3      F(μ) ← encode-Basic(RDD, Σ^R, cons, μ)
4      while TRUE do
5          assignment ← consult-SAT-Solver(F(μ))
6          if assignment ≠ UNSAT then
7              π ← extract-Solution(assignment)
8              collisions ← validate-Plans(π)
9              if collisions = ∅ then
10                 while TRUE do
11                     nogoods ← validate-Cost(π, ξ)
12                     if nogoods = ∅ then
13                         return (π, ∅, UNDEF, UNDEF)
14                     F(μ) ← F(μ) ∪ nogoods
15                     assignment ← consult-SAT-Solver(F(μ))
16                     if assignment = UNSAT then
17                         (μ_next, ξ_next) ←
                             calc-Next-Bounds(μ, ξ, cons, RDD)
18                         return (UNSAT, cons, μ_next, ξ_next)
19                     π ← extract-Solution(assignment)
20             else
21                 for each (m_i × m_j) ∈ collisions where
                     m_i = (a_i, (u_i, v_i), [t_i^0, t_i^+)) and
                     m_j = (a_j, (u_j, v_j), [t_j^0, t_j^+)) do
22                     F(μ)←F(μ)∧(¬E^{t_i^0,t_i^+}_{u_i,v_i}(a_i)∨¬E^{t_j^0,t_j^+}_{u_j,v_j}(a_j))
23                     ([τ_i^0, τ_i^+); [τ_j^0, τ_j^+)) ←
                         resolve-Collision(m_i,m_j)
24                     cons ← cons ∪ {[(a_i, (u_i, v_i), [τ_i^0, τ_i^+));
                         (a_j, (u_j, v_j), [τ_j^0, τ_j^+))]}
25             RDD ←build-Sparse-RDDs(Σ^R, cons, μ)
26             F(μ) ← augment-Basic(RDD, Σ^R, cons)
27     (μ_next, ξ_next) ← calc-Next-Bounds(μ, ξ, cons, RDD)
28     return (UNSAT, cons, μ_next, ξ_next)
```

IV. Experimental Evaluation

We integrated sparse RDDs in a C++ implementation of SMT-CBS$^{\mathcal{R}}$ [1]. The competitive evaluation has been done with respect to original SMT-CBS$^{\mathcal{R}}$ using full RDDs and CCBS, a search-based algorithm. SMT-CBS$^{\mathcal{R}}$ was implemented on top of the Glucose 3 SAT solver [21].

Benchmarks from the `movinai.com` collection [23] have been used. In each benchmark, we interconnected cells using the 2^K-neighborhood [24] for $K = 3, 4, 5$ - the same style of generating benchmarks as used in [10]. Instances consisting of k robots were generated by taking first k robots from the *random* scenario files accompanying each benchmark on `movinai.com`. Having 25 scenarios for each benchmark this yields to 25 instances per number of robots.

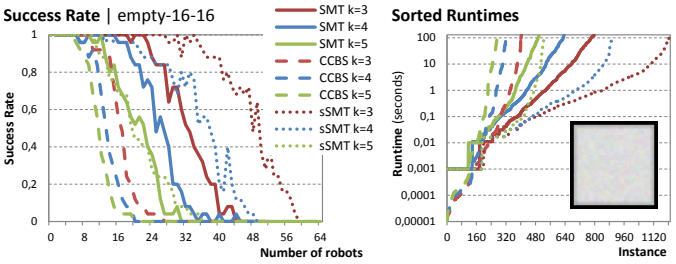

Fig. 4. Comparison of Sparse-SMT-CBS$^{\mathcal{R}}$, SMT-CBS$^{\mathcal{R}}$ and CCBS on `empty-16-16` (small benchmark).

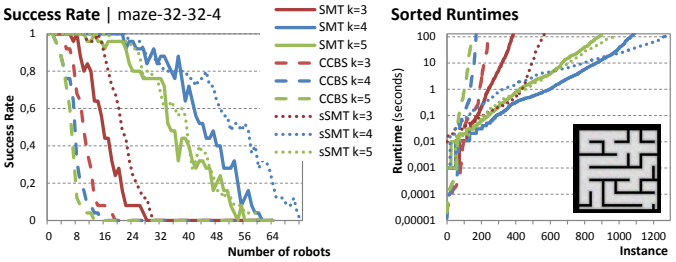

Fig. 5. Comparison of Sparse-SMT-CBS$^{\mathcal{R}}$, SMT-CBS$^{\mathcal{R}}$ and CCBS on `maze-32-32-4` (medium benchmark).

Part of the results obtained in our experimentation is presented here [2]. For each presented benchmark we show *success rate* as a function of the number of robots. That is, we calculate the ratio out of 25 instances per number of robots where the tested algorithm finished under the timeout of 120 seconds. In addition to this, we also show the concrete runtimes sorted in the ascending order. We used three categories of benchmarks according to the size of the map: (i) small, (ii) medium, and (iii) large. Results for one selected representative benchmark from each category are shown in Figures 4, 5, and 6.

Throughout all benchmarks SMT-CBS$^{\mathcal{R}}$ with sparse RDDs dominates over the other two tested algorithms. The dominance of sparse RDDs is surprisingly most visible in small

[1]To enable reproducibility of presented results we provide complete source code of our solver in our github repository: https://github.com/surynek/boOX.

[2]All experiments were run on system consisting of Xeon 2.8 GHz cores, 32 GB RAM, running Ubuntu Linux 18.

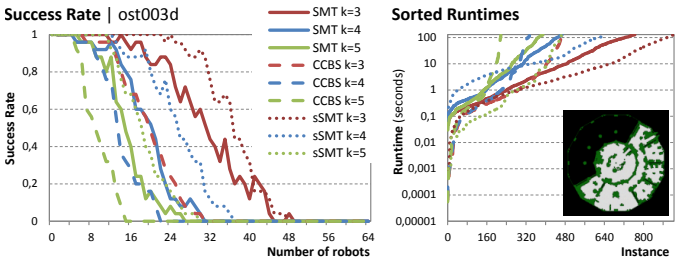

Fig. 6. Comparison of Sparse-SMT-CBS$^{\mathcal{R}}$, SMT-CBS$^{\mathcal{R}}$ and CCBS on `ost003d` (large benchmark).

sized maps however the improvement over full RDDs is observable across all setups.

V. Conclusion

Sparsification of real-time decision diagrams (RDDs) represents new level of laziness in compilation-based algorithms for continuous multi-robot path planning. We have shown that even in its vanilla variant where sparsification is done according to a simple heuristic, in which the growing number of collision avoidance constraints are considered, the technique provides significant performance gains in the SMT-CBS scheme.

Secondarily, sparse RDDs opens room for integrating more advanced heuristics in compilation-based MRPP$^{\mathcal{R}}$ solving which before this technique represents rather a black-box scheme which internal operation can be hardly influenced.

For the future work we plan to develop more advanced heuristics for path selection into RDDs that for example prefer deconflicting between robots at the level of RDD generation.

Acknowledgments

This work has been supported by GAČR - the Czech Science Foundation, grant registration number 19-17966S.

References

[1] D. Silver, "Cooperative pathfinding," in *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*. AAAI Press, 2005, pp. 117–122.

[2] M. R. K. Ryan, "Graph decomposition for efficient multi-robot path planning," in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007, pp. 2003–2008.

[3] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[4] R. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, T. Walsh, Ed. IJCAI/AAAI, 2011, pp. 294–300.

[5] J. Yu and S. M. LaValle, "Planning optimal paths for multiple robots on graphs," in *2013 IEEE International Conference on Robotics and Automation, 2013*. IEEE, 2013, pp. 3612–3617.

[6] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, "A constraint store based on multivalued decision diagrams," in *Principles and Practice of Constraint Programming - CP 2007, Proceedings*, ser. Lecture Notes in Computer Science, vol. 4741. Springer, 2007, pp. 118–132.

[7] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 195, pp. 470–495, 2013.

[8] P. Surynek, A. Felner, R. Stern, and E. Boyarski, "Efficient SAT approach to multi-agent path finding under the sum of costs objective," in *ECAI 2016 - 22nd European Conference on Artificial Intelligence*, ser. Frontiers in Artificial Intelligence and Applications, vol. 285. IOS Press, 2016, pp. 810–818.

[9] T. T. Walker, N. R. Sturtevant, and A. Felner, "Extended increasing cost tree search for non-unit cost domains," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, J. Lang, Ed. ijcai.org, 2018, pp. 534–540. [Online]. Available: https://doi.org/10.24963/ijcai.2018/74

[10] A. Andreychuk, K. S. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*. ijcai.org, 2019, pp. 39–45.

[11] P. Surynek, "On satisfisfiability modulo theories in continuous multi-agent path finding: Compilation-based and search-based approaches compared," in *Proceedings of the 12th International Conference on Agents and Artificial Intelligence, ICAART 2020*. SCITEPRESS, 2020, pp. 182–193.

[12] J. Li, P. Surynek, A. Felner, H. Ma, T. K. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 7627–7634.

[13] D. Ratner and M. K. Warmuth, "Nxn puzzle and related relocation problem," *J. Symb. Comput.*, vol. 10, no. 2, pp. 111–138, 1990.

[14] J. Yu and S. M. LaValle, "Optimal multi-robot path planning on graphs: Structure and computational complexity," *CoRR*, vol. abs/1507.03289, 2015.

[15] M. Phillips and M. Likhachev, "SIPP: safe interval path planning for dynamic environments," in *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*. IEEE, 2011, pp. 5628–5635.

[16] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artif. Intell.*, vol. 219, pp. 40–66, 2015.

[17] P. Surynek, "Sparsification for fast optimal multi-robot path planning in lazy compilation schemes," *CoRR*, vol. abs/2103.04496, 2021, to appear in IROS 2021. [Online]. Available: https://arxiv.org/abs/2103.04496

[18] G. Gange, D. Harabor, and P. J. Stuckey, "Lazy CBS: implicit conflict-based search using lazy clause generation," in *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018*. AAAI Press, 2019, pp. 155–162.

[19] E. Lam, P. L. Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*. ijcai.org, 2019, pp. 1289–1296.

[20] P. Surynek, "Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 1177–1183.

[21] G. Audemard and L. Simon, "On the glucose SAT solver," *Int. J. Artif. Intell. Tools*, vol. 27, no. 1, pp. 1 840 001:1–1 840 001:25, 2018.

[22] H. A. Kautz and B. Selman, "Unifying sat-based and graph-based planning," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*. Morgan Kaufmann, 1999, pp. 318–325.

[23] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, 2012.

[24] N. Rivera, C. Hernández, and J. A. Baier, "Grid pathfinding on the 2k neighborhoods," in *Proceedings of AAAI 2017*, 2017, pp. 891–897.