A Decomposed Symbolic Approach to Reactive Planning

Seung H. Chung and Brian C. Williams

Artificial Intelligence and Space Systems Laboratories
Massachusetts Institute of Technology
77 Massachusetts Ave. 37-346
Cambridge, MA 02139
{chung, williams}@mit.edu

Abstract. Autonomous systems in uncertain dynamic environments must reconfigure themselves in response to unanticipated events and goals in real-time. We present an approach to reactive configuration planning based on the principle of decomposition. Reactive plans are susceptible to exponential state space explosion. We address this problem through transition-based decomposition by generating compact *decomposed goal-directed plans*. We further minimize state explosion by adopting a symbolic representation based on Ordered Binary Decision Diagrams. We demonstrate our reactive planner on representative spacecraft subsystem models.

1 Introduction

Recent failures in NASA's Mars exploration program point to the need for increased autonomous response in spacecraft. The presumed cause of failure for the Mars Polar Lander (MPL) Mission [1] provides a relevant example. During the final stage of MPL's descent to the Martian surface, one sensor wrongfully signaled the landing of the spacecraft. As a result, the descent engines were prematurely shut off, causing MPL to crash into the surface. During this incident, no communication was possible between MPL and ground control. Even if it were, the outcome would likely have been inevitable due to the communication time delay: when Mars is closest to Earth, commands from the ground take at least 12 minutes to reach the spacecraft. Thus, an onboard reactive system is necessary to autonomously respond to anomalies.

1.1 Motivation for Tractable Reactive Planning

While general-purpose onboard planners could be used for autonomous reconfiguration, due to the PSPACE-complete nature of planning problems, real-time response cannot be guaranteed. In time-critical situations, such as the MPL landing scenario, late response could be disastrous to the mission. Reactive planning is an approach that guarantees real-time response. A reactive planner precompiles a plan offline for all possible situations, and then executes the plan online.

In general, a reactive planner may not be able to optimize resource utilization. However, the irreversibility associated with the use of nonrenewable resources requires careful deliberation to ensure system safety and mission success:

Requirement 1. *A reactive planner shall consider only reversible control actions, unless the effect is to repair failures* [2].

One of the early approaches to reactive planning is universal planning, first introduced by [3]. Though a universal plan can react to a nondeterministic environment, it cannot react to rapidly changing goals. Furthermore, [4] pointed out the intractability of universal planning due to the exponential state space explosion problem. Thus, a new reactive planning approach is necessary.

1.2 Handling State Explosion through Decomposition

The method of divide-and-conquer is a well known effective approach to solving problems. Based on this principle, we have developed a new transition-based decomposition method for reactive planning. Though this method is unrelated to the structural decomposition used in constraint satisfaction problems (CSP) [5], the contribution of our decomposition method to planning is analogous to that of the structural decomposition methods for CSP.

CSPs are known to be NP-complete, but [6] has shown that a CSP with a tree-structured constraint graph is solvable in linear time. Similarly, [2] have shown that if a planning problem has an acyclic dependency, then the problem can be solved within a state space that grows only linear in the number of state variables. For CSPs that do not have tree-structured constraint graph, Dechter and Pearl have shown that the constraint graphs of those problems can be transformed into tree-structured graphs using a tree decomposition technique [7]. In our approach, for planning problems with cyclic transition dependency graph (TDG), we use transition-based decomposition to transform the cyclic TDG to an acyclic TDG. As a result, even planning problems with cyclic TDG can be solved within a state space that grows approximately linear in the number of state variables.

1.3 Handling State Explosion through Symbolic Representation (OBDD)

Through transition-based decomposition, our reactive planner divides a problem into a set of subproblems. While transition-based decomposition addresses the state explosion problem at the global level, we also address this issue at the subproblem level by adopting a symbolic representation.

The logic synthesis and model checking communities have been using Ordered Binary Decision Diagrams (OBDD) [8] for compact state space encoding. An OBDD-based model checking technique has proven particularly successful in dealing with the state explosion problem [9]. Recognizing the similarities between model checking and planning, [10] introduced a new universal planning technique that takes advantage of the OBDD. Since then, several OBDD-based universal planning algorithms have been introduced for operating within nondeterministic domains [11–13]. In our approach, we also take advantage of OBDD, but unlike the universal planners, our reactive planner generates *goal-directed plans* (GDP) that can react to the nondeterministic environment as well as rapidly changing goals. We generate these GDPs for each decomposed subproblem such that the resulting *decomposed goal-directed plan* (DGDP) conforms to Requirement 1 while guaranteeing real-time response.

2 Spacecraft Communication System Example

Throughout the paper, we will use a model of a simplified spacecraft communication system (Figure 1) to present our decomposed symbolic approach to reactive planning. Figure 1 depicts the direction of signal flow among components. The computer sends data to be transmitted through the bus control. When the data is received, the bus control routes it to the transmitter. The transmitter receives the data and generates the corresponding signal. The signal is amplified by the amplifier and is finally transmitted through the antenna. The computer is also responsible for controlling the devices: it may command either the transmitter or amplifier to be turned on or off. Again, these commands are sent to the appropriate devices via the bus control.

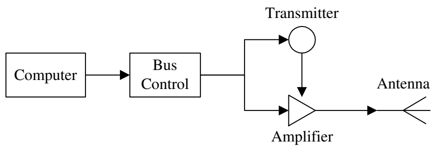

Fig. 1. Simplified spacecraft communication system.

2.1 Modelling Behavior with Concurrent Automata

We model a system of concurrently operating components by a set of *concurrent automata*. Figure 2 illustrates the concurrent automata of a transmitter and an amplifier. The transitions between states are conditioned on commands (e.g. $cmd_T = off$) and states of other automata. For instance, the amplifier must be turned off ($A = off$) before we can command the transmitter on or off. This particular condition is necessary for the safety of the system, as the process of switching the transmitter on or off may generate a transient signal spike that could damage the amplifier. For the same reason, the transmitter must be turned on before the amplifier can be turned on. We define concurrent automata formally as follows:

Definition 1. *A set of* concurrent automata $\mathcal{CA} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}, \ldots, \mathcal{A}^{(n)}\}$ *is composed of concurrently operating finite automata. Each concurrent automaton* $\mathcal{A}^{(i)}$ *is a 3-tuple* $\langle Q^{(i)}, \Sigma^{(i)}, \delta^{(i)} \rangle$*, where* $Q^{(i)}$ *is a finite set of states,* $\Sigma^{(i)}$ *is a finite set of inputs (either commands or states of other concurrent automata) and* $\delta^{(i)} : Q^{(i)} \times \Sigma^{(i)} \to Q^{(i)}$ *is a transition function.*

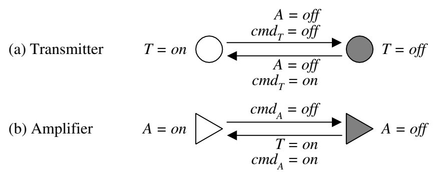

Fig. 2. Concurrent automata of a transmitter and an amplifier. Idle transitions are omitted for clarity.

2.2 Representing $\mathcal{CA}$ Symbolically

For compactness, we encode the concurrent automata in an OBDD representation. In this representation, state s of concurrent automaton i is defined by a vector of $\log_2(|Q^{(i)}|)$ distinct Boolean variables, where $|Q^{(i)}|$ is the number of elements in $Q^{(i)}$. Similarly, the input a is represented as a vector of Boolean variables. The transition relation for concurrent automaton i is $R^{(i)} : Q^{(i)} \times \Sigma^{(i)} \times Q^{(i)} \to \mathcal{B}$, where $\mathcal{B}$ is a set of Boolean values and $R^{(i)}(s, a, s') = (s' \in \delta^{(i)}(s, a))$, and s' indicates the state at the next time step. For example, the transition relation $R^{(A)}$ of the amplifier A is as follows:

$$\begin{aligned}
\left[((A = \mathit{off}) \wedge \neg((T = \mathit{on}) \wedge (\mathit{cmd}_A = \mathit{on}))) \Rightarrow (A' = \mathit{off})\right] \wedge \\
\left[((A = \mathit{off}) \wedge (T = \mathit{on}) \wedge (\mathit{cmd}_A = \mathit{on})) \Rightarrow (A' = \mathit{on})\right] \wedge \\
\left[((A = \mathit{on}) \wedge (\mathit{cmd}_A = \mathit{off})) \Rightarrow (A' = \mathit{off})\right] \wedge \\
\left[((A = \mathit{on}) \wedge \neg(\mathit{cmd}_A = \mathit{off})) \Rightarrow (A' = \mathit{on})\right]
\end{aligned}$$

Figure 3(a) illustrates the OBDD representation of the transition $((A = \mathit{on}) \wedge (\mathit{cmd}_A = \mathit{off})) \Rightarrow (A' = \mathit{off})$. Figure 3(b) shows the result of conjoining the OBDDs of the transitions into the transition $R^{(A)}$[1]. Each node of an OBDD represents a Boolean variable, and the dotted and the solid outgoing edges respectively represent false and true evaluations of the Boolean variable. The terminal nodes 1 and 0 represent the evaluation of the Boolean function (i.e. OBDD) where each path from the root to a terminal evaluates to 1 for true or 0 for false. In Figure 3(b), all paths that lead to false have been omitted for simplicity. One of the benefits of using OBDDs to represent transition relations is in relative compactness of OBDDs. [12] shows that the size of an OBDD does not necessary depend on the number of states, but rather on the structure of the information the OBDD encodes.

[1] In this example, we assume cmd_T and cmd_A are on or off at all times.

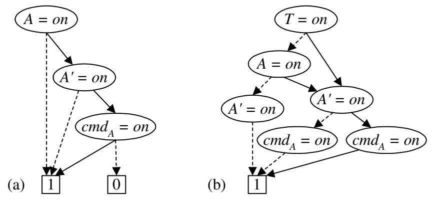

Fig. 3. OBDD representation of (a) $((A = \mathit{on}) \wedge (\mathit{cmd}_A = \mathit{off})) \Rightarrow (A' = \mathit{off})$ and (b) amplifier transition relation $R^{(A)}$.

3 Subgoal Serialization through Transition-based Decomposition

A set of subgoals are serializable if and only if a goal can be partitioned into a set of subgoals that can be solved sequentially to achieve the goal [14]. For example, consider the driver and valve shown in Figure 4. The driver is a device that commands the valve open or closed. Thus, the driver must be on $(D = \mathit{on})$, before the valve can be commanded open or closed. Presume that the current state of the driver and valve system is $(D = \mathit{off}, V = \mathit{closed})$ and the goal state to achieve is $(D = \mathit{off}, V = \mathit{open})$. In this case, we do not have to figure out how to achieve $(D = \mathit{off})$ and $(V = \mathit{open})$ simultaneously. Rather, we can figure out how to achieve $(V = \mathit{open})$ first. Once $(V = \mathit{open})$ has been achieved, we can then figure out how to achieve $(D = \mathit{off})$, without worrying about potential impact on the $(V = \mathit{open})$ subgoal. Hence, the subgoals are serializable.

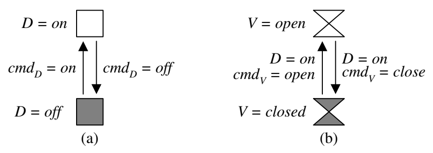

Fig. 4. Concurrent automata for (a) Driver, and (b) Valve.

[2] recognized that a set of subgoals are serializable if the transition dependency graph (TDG) of a system is acyclic, where TDG of a concurrent automata is formally defined as follows:

Definition 2. *A* transition dependency graph $\mathcal{G}$ *of* $\mathcal{CA}$ *is a directed graph whose vertices are the concurrent automata* $\{\mathcal{C}\}$. $\mathcal{G}$ *contains a directed edge from vertex* $\mathcal{C}^{(i)}$ *to vertex* $\mathcal{C}^{(j)}$, *if* $\mathcal{C}^{(i)}$ *occurs in the antecedent (precondition) of one of* $\mathcal{C}^{(j)}$*'s transitions.*

For the driver and valve, the valve's ability to open or close depends on the state of the driver. The driver, however, does not depend on the valve, so we can change the driver state without affecting the valve state. Hence, the dependency relationship is acyclic.

For a system with a cyclic TDG, we can transform it into an acyclic graph through transition-based decomposition. For example, TDG of the communication system is cyclic, as shown in Figure 5. However, if we group the transmitter and the amplifier, and consider them as a single vertex in TDG, the resulting graph is acyclic. We recognize that a set of cyclic vertices in TDG directly corresponds to a strongly connected component (SCC) of the TDG.

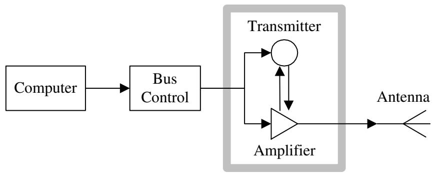

Fig. 5. Cyclic transition dependency graph of a spacecraft communication subsystem.

4 Goal-directed Plan

With the TDG decomposed into a set of SCCs, we can generate a GDP for each SCC individually. As the automata within a SCC have cyclic dependency, we must consider the concurrency and interdependence of the automata. With this in mine, we first compose the automata within a SCC into a single automaton. Then, we generate a GDP based on the composed automaton.

4.1 Composing Automata

Continuing with the transmitter/amplifier example, we want to construct a single automaton that represents both components, as shown in Figure 6. Notice that one transition seems missing, the transition from $(T = on, A = \mathit{off})$ to $(T = \mathit{off}, A = on)$. According to the model shown in Figure 2, such a transition may occur if the transmitter is commanded off $(cmd_T = \mathit{off})$ and the amplifier is commanded on $(cmd_A = on)$ simultaneously. In controlling concurrent devices, however, such synchronized control cannot be guaranteed; in fact, such commanding is nearly impossible, and taking such an action could be hazardous, as the amplifier may be damaged if $(cmd_A = on)$ precedes $(cmd_T = \mathit{off})$ even by a fraction of a second. Thus, before composing automata, we modify the transition relation for each automaton to avoid such hazards. If a transition is conditioned on the state of another automaton, we require that the state condition be true before and after the transition occurs.

For example, consider the amplifier's transition from off to on:

$$((A = \mathit{off}) \wedge (T = on) \wedge (cmd_A = on)) \Rightarrow (A' = on)$$

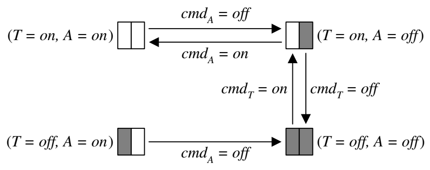

Fig. 6. Automaton of composed transmitter and amplifier automata. Idle transitions are omitted for clarity.

The transition relies on the transmitter being on ($T = on$). Thus, we modify the transition to guarantee that the transmitter is on before and after:

$$((A = \mathit{off}) \wedge (T = on) \wedge (T' = on) \wedge (cmd_A = on)) \Rightarrow (A' = on)$$

With the modified transition relations, composing the concurrent automata into a single automaton is trivial. The composed transition relation R_{SCC} of a SCC is

$$R_{SCC} = \bigwedge_{\mathcal{C}^{(i)} \in SCC} R^{(i)}$$

4.2 Generating the Goal-directed Plan

A GDP is comprised of a set of goal-directed rules, where a goal-directed rule is a 3-tuple $\langle s, a, s' \rangle$. A goal-directed rule can be interpreted as "if the current state is s and the goal state is s', execute a". Figure 7 is a tabular representation of the goal-directed plan for the transmitter/amplifier SCC. Each entry in the table corresponds to a goal-directed rule. While a in this GDP is only composed of commands, a may in general contain states of other automata that precede the SCC in the dependency ordering. For example, one of the goal-directed rules for the valve is

$$\langle (V = open), (D = on, cmd_V = close), (V = closed) \rangle .$$

($D = on$) is an *intermediate subgoal* that must be achieved before we can command the valve closed.

We generate the GDP by iteratively searching the state space in parallel, backward, and breadth-first manner. With OBDDs, states within the search space do not have to be enumerated; instead, we can generate goal-directed rules of all goals and initial states simultaneously, thus "in parallel". The search method is also characterized as a "backward search", as the GDP is generated by searching for the states that can reach the goal, instead of searching for the goals that can be reached from the current state. Of the goal-directed rules, we generate the one-step rules (i.e. goal-directed rules with goals that can be achieved in a single transition) fist. In Figure 7, one-step rules are those with "(1)" next to the commands. Notice that one-step rules correspond directly to

Current State	Goal State			
	T=on,A=on	T=on,A=off	T=off,A=off	T=off,A=on
T=on,A=on	*idle*	cmd_A=off(1)	cmd_A=off(2)	*failure*
T=on,A=off	cmd_A=on(1)	*idle*	cmd_T=off(1)	*failure*
T=off,A=off	cmd_T=on(2)	cmd_T=on(1)	*idle*	*failure*
T=off,A=on	cmd_A=off(3)	cmd_A=off(2)	cmd_A=off(1)	*idle*

Fig. 7. Goal-directed plan for the transmitter/amplifier system. The number next to each command represents the total number of steps necessary to achieve the goal.

the transitions in transition relation. Next, we generate two-step rules, labelled "(2)" in Figure 7. We continue this process until the fixed-point is reached, thus "breadth-first". In general, the fixed-point of the iterative search is defined by the width of the transition graph of the automaton. In our transmitter/amplifier example, the fixed-point is reached after two iterations (i.e. after the three-step rules are generated). The algorithm for generating the GDP is as follows:

Algorithm 1 ComputeGDP(T)

```
1: $oldPlan \leftarrow \emptyset$
2: $newPlan \leftarrow T$
3: while $oldPlan \neq newPlan$ do
4:     $oldPlan \leftarrow newPlan$
5:     $newPlan \leftarrow oldPlan \cup$ ComputeNextStepRules($T, oldPlan$)
6: return $newPlan$
```

The algorithm ComputeGDP takes the transition relation T of an automaton as its input. As we have discussed, the one-step rules are exactly the transition relation as reflected in line 2 of ComputeGDP. In lines 3–5, it iteratively searches for two-step rules, three-step rules, etc. while adding them to the newPlan. The procedure exits once the fixed-point is reached (line 3), and returns the plan (line 6).

In line 5, ComputeNextStepRules(T,oldPlan) generates n-step rules when the oldPlan contains all rules of less than n-steps. Assume that a relation $s_i \wedge a_j \Rightarrow s'_k$ is in the transition relation T and an $(n-1)$-step rule $\langle s_k, a_l, s'_m \rangle$ is in the old goal-directed plan oldPlan. Then, $\langle s_i, a_j, s'_m \rangle$ is one of the valid n-step rules returned by ComputeNextStepRules(T,P). For example, from the 1-step rule

$$\langle (T = on, A = off), (cmd_T = off), (T' = off, A' = off) \rangle$$

and the transition relation

$$((T = on, A = on) \wedge (cmd_A = off)) \Rightarrow (T' = on, A' = off)$$

the 2-step rule

$$\langle (T = on, A = on), (cmd_A = off), (T' = off, A' = off) \rangle$$

can be deduced.

Formally: COMPUTENEXTSTEPRULES(T,P) generates a set of n-step goal-directed rules $\langle s, a, s' \rangle$, where T is a transition relation and P is a goal-directed plan with only m-step rules, where $m < n$. Each rule $\langle s_i, a_j, s'_k \rangle$ is restricted such that $s'_l \subseteq (\mathtt{T} \wedge s_i \wedge a_j)$, $\langle s_l, a_m, s'_k \rangle \in \mathtt{P}$, and $\neg\exists a. \langle s_i, a, s'_k \rangle \in \mathtt{P}$.

$s'_l \subseteq (\mathtt{T} \wedge s_i \wedge a_j)$ states that s'_l must be reachable from state s_i through input a_j. The restriction $\neg\exists a. \langle s_i, a, s'_k \rangle \in \mathtt{P}$ says that $\langle s_i, a_j, s'_k \rangle$ cannot be a new goal-directed rule if a rule for the current state s_i and the goal state s'_k already exists in the plan P. With this restriction, the resulting GDP is guaranteed to be optimal, where an optimal plan is defined as a plan with the shortest control sequence. For example, while

$$\langle (T = on, A = off), (cmd_T = off), (T' = on, A' = on) \rangle$$

is a 3-step rule, it is not a valid rule since the optimal 1-step rule already exists in the plan:

$$\langle (T = on, A = off), (cmd_A = on), (T' = on, A' = on) \rangle$$

The algorithm for COMPUTENEXTSTEPRULES(T,P) is shown in algorithm ??. This algorithm leverages the OBDD representation to efficiently search the state space, without enumeration.

Algorithm 2 COMPUTENEXTSTEPRULES(T, P)

```
1: $nextPlan \leftarrow T_{[sTemp/s']} \wedge \exists a.P_{[sTemp/s]}$
2: $optimalNextPlanWithNoCmd \leftarrow \exists a.nextPlan - \exists a.P$
3: **return** $nextPlan \wedge optimalNextPlanWithNoCmd$
```

Line 1 of COMPUTENEXTSTEPRULES(T,P) computes all next step goal-directed rules including the non-optimal ones[2]. Line 2 determines which rules are the valid (i.e. optimal) rules. Finally, line 3 returns only those rules that are valid.

5 Decomposed Goal-directed Plan

Once the TDG is made acyclic, we can generate a GDP for each SCC successively, instead of generating a single GDP for the whole $\mathcal{CA}$. This set of GDPs for all SCCs in the system is called a *decomposed goal-directed plan* (DGDP). The advantage of this composition is that the footprint of the DGDP is much smaller than a single GDP for the full $\mathcal{CA}$. For example, let us assume that the number of concurrent automata, $|\mathcal{CA}|$, is n, and the average number of states per automaton, $|Q|$, is m. If we generate a single GDP for $\mathcal{CA}$, the number of states in GDP is exponential in $|\mathcal{CA}|$, $O(m^n)$. If the maximum number of automata

[2] [sTemp/s] symbolizes the replacement of variable s with variable sTemp.

in an SCC is w, however, the number of states in the corresponding DGDP is only $O(l \cdot m^w)$, where l is the total number of SCCs. Thus, even if the size of a $\mathcal{CA}$ grows, as long as m and w remains constant, the size of the corresponding DGDP grows only linearly in l. The algorithm for generating DGDP is of $\mathcal{CA}$ as follows:

Algorithm 3 ComputeDGDP(n, R, q)

```
1: $revReachAncs \leftarrow \emptyset$
2: for $i = 0$ to $(n-1)$ do
3:     $allwdR \ \leftarrow R[i] \wedge revReachAncs$
4:     $DGDP[i] \leftarrow$ ComputeGDP$(allwdR)$
5:     $revReachAncs \leftarrow revReachAncs \cup$ ComputeRRS$(R[i], q[i])$
6: return $DGDP$
```

where `n` is the number of composed automata (i.e. SCCs), `R` is an array of transition relations of the composed automata sorted in dependency order (i.e. inverse depth-first order of TDG), and `q` is an array of current states of the composed automata, also in dependency order. Lines 2–5 successively generates the GDP of each SCC in dependece order, storing an array of GDPs in `DGDP` (line 4). A GDP is computed from a subset of the SCC transition relation, `allwdR`, where the subset is restricted to the transitions whose antecedents (subgoals in GDP) are reversibly reachable from the current state `q`. This restriction guarantees the aforementioned requirement 1. In line 5, the reversibly reachable states of the `i`-th SCC are generated and added to the set of reversibly reachable ancestor states `revReachAncs` to be used in the next iteration.

6 DGDP Execution

Figure 8 shows a DGDP for a driver and a valve. We execute DGDP in inverse dependency order (e.g. the valve then the driver). For example, let us assume the driver and valve are off and closed, respectively, and the goal is to turn off the driver and open the valve. First, we must attempt to open the valve, according to the inverse dependency order. To switch the valve open from the closed position, the driver must be on and the valve must be commanded open as shown in Figure 8. Here, $(D = on)$ is a subgoal that must be achieved before executing the command $(cmd_V = open)$. As the driver is currently off, we determine how to turn the driver on by looking up the driver plan in Figure 8. According to the plan, we simply command the driver on $(cmd_D = on)$. Once the driver is turned on, then we can command the valve to open $(cmd_V = open)$. Once the valve is opened, then the drive can be turned off again. [2] discuss DGDP execution algorithm in detail. The incremental nature of the algorithm allows for robust execution that immediately responds to failures or sudden changes in goals.

Current State	Goal State	
	D=*on*	D=*off*
D=*on*	*idle*	cmd_D=*off*
D=*off*	cmd_D=*on*	*idle*

Current State	Goal State	
	D=*on*	D=*off*
V=*open*	*idle*	D=*on* cmd_V=*close*
V=*closed*	D=*on* cmd_V=*open*	*idle*

Fig. 8. Factored goal-directed plan for a valve driver (left) and a valve (right).

7 Conclusion

Our decomposed symbolic approach to reactive planning is novel in two ways. First, it leverages transition-based decomposition to eliminate the state space explosion problem in reactive planning. When transition-based decomposition is used to solve a problem, the complexity of the problem becomes linear in the size of the SCCs instead of being exponential in the size of $\mathcal{CA}$. As long as the size of the SCCs remains relatively small, the problem remains tractable. Second, we incorporate the use of OBDDs into reactive planning, which gives us two distinct advantages: (1) we can search the state space without the need to enumerate the states, and (2) we can take advantage of the OBDD's compact state space encoding capability.

8 Acknowledgments

This work was supported in part by NASA's Cross Enterprise Technology Development program under contract NAG2-1466, DARPA MOBIES program under contract F33615-00-C-1702, and NASA Graduate Student Research Program Fellowship.

References

1. Casani, J., Whetsler, C., Albee, A., Battel, S., Brace, R., Burdick, G., Burr, P., Dippoey, D., Lavell, J., Leising, C., MacPherson, D., Menard, W., Rose, R., Sackheim, R., Schallenmuller, A.: Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions. Technical Report JPL D-18709, Jet Propulsion Laboratory, California Institute of Technology (2000)
2. Williams, B.C., Nayak, P.P.: A Reactive Planner for a Model-based Executive. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Japan (1997)
3. Schoppers, M.J.: Universal Plans for Reactive Robots in Unpredictable Environments. In: Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI'87). Volume 2., Milan, Italy (1987) 1039–1046
4. Ginsberg, M.L.: Universal Planning: An (Almost) Universally Bad Idea. AI Magazine **10** (1989) 40–44

5. Gottlob, G., Leone, N., Scarcello, F.: A Comparison of Structural CSP Decomposition Methods. In Dean, T., ed.: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), Stockholm, Sweden (1999) 394–399
6. Freuder, E.C.: A Sufficient Condition for Backtrack-Bounded Search. Journal of the ACM (JACM) **32** (1985) 755–761
7. Dechter, R., Pearl, J.: Tree Clustering for Constraint Networks. Artificial Inteligence **38** (1989) 353–366
8. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **C-35** (1986) 677–691
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang:, L.J.: Symbolic Model Checking: 10^{20} States and Beyond. Information and Computation **98** (1992) 142–170
10. Cimatti, A., Giunchiglia, F., Giunchiglia, E., Traverso, P.: Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$. In: Proceedings of the Fourth European Conference on Planning (ECP'97), Toulouse, France (1997)
11. Cimatti, A., Roveri, M., Traverso, P.: Strong Planning in Non-Deterministic Domains via Model Checking. In: Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS'98), Pittsburgh, Pennsylvania (1998)
12. Cimatti, A., Roveri, M., Traverso, P.: Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In: Prodeedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98), Madison, Wisconsin (1998)
13. Jensen, R.M.: OBDD-based Universal Planning in Multi-Agent, Non-Deterministic Domains. Master's thesis, Technical University of Denmark (1999)
14. Korf, R.E.: Planning as Search: A Quantitative Approach. Artificial Intelligence **33** (1987) 65–68