Utilization of Parallelization for Fast Computation of Controlled Invariant Sets of Transition Systems

Ezra Altabet, ealtabet@umich.edu

***Abstract*— Controlled Invariant sets (CISs) are often an appealing approach to robot path planning since they guarantee system safety. However, the computational complexity needed to compute CISs is often too resource demanding to be practical in real-world applications. [1] proposed a method for computing the maximal CIS of a multi-robot transition system that guarantees finite-step convergence. They then demonstrate their method on an example 7-node, 3-robot, transition system. I have applied parallelization techniques to the transition system formation step of their algorithm such that larger, more practical transition systems can be computed in a relatively short period of time. I then propose options for further computational efficiency and development of transition systems that more closely apply to real-world scenarios. The code for this project is available at `https://github.com/Ealt1/snake_repo_parallelized/tree/master`**

I. Introduction

A controlled invariant set (CIS) in the context of a control system is a subset of the system's state space by which for any trajectory starting within the set there exists a control law that will ensure the trajectory remains within the set. Often the controlled invariant set is used as a way of guaranteeing "safety" of the trajectory. I first define X_{safe} to be the set of all states of the control system that are permittable. It can be easily be imagined that in a given control system it is possible that there exists states where although state x_k is within X_{safe}, x_{k+1} will be forced out of X_{safe} and the system will enter an undesirable state. To remedy this, a controlled invariant subset of X_{safe}, C, is created, representing all states x_k in C such that there exists a control law that guarantees x_{k+1} will remain in C and, therefore, also remain within X_{safe}.

For linear systems the computation of CISs is relatively straightforward, so much so that Anevlavis and Tabuada developed an algorithm to compute CISs for linear systems in two moves [2]. However, linear systems are generally poor at representing real-world control systems and there is no such formulaic approach for nonlinear systems. The most common approach utilizes value iteration (more in II) but this approach is often computationally complex and generally has no guarantee of convergence. However it has been demonstrated that convergence in finite steps can be guaranteed for the case of finite transition systems [1].

In this paper, I explore whether construction of CISs on multi-robot transition systems can be used to aid policy development for multi-robot high-level path planning in time sensitive scenarios. The primary advantage of this approach is that, in addition to guaranteeing a policy that will yield mission success, all computations are complete ahead of time. Therefore, at mission start, the algorithm simply has to select the appropriate policy associated with the distribution at mission start, and can immediately implement the policy towards mission path planning. The disadvantage, as will be demonstrated, is that there is often a severe trade off between transition system CIS computation time and the granularity of the environment model. This means that assumptions made about the environment to speed up computation time can, in many cases, lead to overly conservative policies.

I will demonstrate how parallelization can be implemented to minimize CIS computation without sacrificing environment granularity. I will provide a feasibility report that quantifies the advantages of parallelization in CIS computation and provide recommended applications for the current algorithm. I will then provide the limitations of the current algorithm and a recommended research path to further improve computational time and provide high-resolution representations of the environment.

II. Scenario Explanation and Personal Contributions

Liren Yang's work was demonstrated using a simple real-world robot scenario. This same scenario will be used to demonstrate the benefits of parallelization. His system consists of a 3-robot system in 7-node environment. One robot is a large vehicle that drive on roads, but cannot drive off-road. The other two robots are small vehicles that can be deployed and can access any node on the map; however, they only have a limited amount of fuel and must return to the large vehicle to refuel. Targets are then placed nondeterministically in the locations only accessible to the small vehicles. These targets must be approached within a certain time or mission failure will be triggered. This generic scenario has multiple real-world applications such as for package acquisition, or search-and-rescue robots, where the target must be reached in a time sensitive manner. We can define the above transition system mathematically as follows:

Define a system $\Sigma = (M, X, U, \mu, \tau)$ such that:

- Map $M = (V, E)$ is a directed graph, where V is the vertex set, and $E \subseteq V \times V$ is the edge set. We will assume that $\forall v \in V \, (v, v) \in E$
- State set X. In this scenario $x \in X = [v_0, v_1, v_2, f_1, f_2, v_g, l_g, l_1, l_2]$ where:
  - $v_0 \in V$, the position of the a large vehicle
  - $v_1, v_2 \in V$, the position of two small vehicles
  - $f_1, f_2 \leq \overline{f}$, the fuel level of the two small vehicles
  - $v_g \in V$, the positoin of the target object to collect
  - $l_g \leq \overline{l_g}$, the remaining life of the target item

  - $l_1, l_2 \leq \bar{l}$, the remaining life of the small vehicles in the unsafe region

  Here, $\overline{f}$ (fuel tank size), $\overline{l_g}$ and $\bar{l}$ (small vehicle's full life) are given constants.
- Control set U. A control input $u = [u_0, u_1, u_2]$ where
  - $u_0 \in V$ is the location of the large vehicle after a movement
  - $u_1, u_2 \in V$ are the locations of the two small vehicles after a movement
- Control map $\mu : X \to 2^U$. A set $\mu(x)$ contains all the allowable control inputs at state x. Particularly, $u \in \mu(x)\, iff$
  - $\forall i \in 0, 1, 2 : (v_i, u_i) \in E$, i.e., the movements of the three vehicles respect the map
  - $\forall i \in 1, 2 : (v_i \neq u_i) \to (f_i > 0)$, i.e. the small vehicles cannot move without fuel.
- Transition map $\tau : X \times U \to 2^X$. Let x be the current state, $u \in \mu(x)$ be the control input, the next state $x^+ \in \tau(x, u) iff$
  - $\forall i \in 0, 1, 2 : v_i^+ = u_i$
  - $\forall i \in 1, 2$
    * $(v_i^+ = v_0^+) \to (f_i^+ = \overline{f})$, i.e.e, large and small vehicles meet, recharge
    * $((v_i^+ \neq v_0^+) \wedge (v_i^+ \neq v_i^+)) \to (f_i^+ = f_i - 1)$, i.e., nontrivial movement, fuel drops
    * $((v_i^+ \neq v_0^+) \wedge (v_i^+ = v_i^+) \wedge (f_i > 0)) \to (f_i^+ = f_i)$, i.e., no movement, fuel unchanged
    * $((v_i^+ \neq v_0^+) \wedge (v_i^+ = v_i^+) \wedge (f_i \leq 0)) \to (f_i^+ = f_i - 1)$, no fuel movement, no movement, f_i drops to negative to keep track of the time spent in the unsafe region
    * $(v_i \in V \sim V_{safe}) = \to (l_i^+ = l_i - 1)$, i.e., the small vehicle's life decreases by one for every time step it spend in the unsafe region
    * $(v_i \in V_{safe}) \to (l_i^+ = \bar{l})$, i.e., the life of the small vehicles reset after coming back to the safe region
  - $((v_g = v_1) \vee (v_g = v_2)) \to ((v_g^+ \in V_{target}) \wedge (l_g^+ = \overline{l_g}))$, i.e., item captures, generate new item in $V_{target} \subseteq V$, reset item life to $\overline{l_g}$
  - $((v_g v_1) \wedge (v_g \neq v_2)) \to ((v_g^+ = v_g) \wedge (l_g^+ = l_g - 1))$, i.e., item not captured

Next, define Specifications Φ

- The large vehicle stay in a set of safe locations

$$V_{safe} \subseteq V : \phi_0 = \Box(v_0 \in V_{safe}) \tag{1}$$

- The target item is collected before it vanishes

$$l_g < 0 : \phi_g = \Box(l_g \geq 0) \tag{2}$$

- A small vehicle gets recharged within at most $\overline{d}$ steps after its fuel level reaches empty:

$$\phi_i = \Box(\bigwedge_{t=0}^{\overline{d}} \bigcirc^t (f_i = 0) \to \bigcirc^{\overline{d}+1}(f_i > 0)) \tag{3}$$

- The small vehicles does not stay in the unsafe region for more than $\bar{l}$ steps:

$$\psi_i = \Box(\bigwedge_{t=0}^{\bar{l}} {}^t(v_i \in V_{safe}) \to^{\bar{l}+1} (v_i \in V_{safe})) \tag{4}$$

The overall specification is defined by:

$$\Phi = \bigwedge_{i=1}^{2} (\phi \wedge \psi_i) \wedge \phi_0 \wedge \phi_g \tag{5}$$

With auxiliary variables and the transitions, the specification can be written as

$$\Phi' = \bigwedge_{i=1}^{2} (\phi' \wedge \psi_i') \wedge \phi_0 \wedge \phi_g \tag{6}$$

where:

$$\phi_i' = \Box(f_i \geq -\overline{d}) \tag{7}$$
$$\psi_i = \Box(l_i \geq 0) \tag{8}$$

To obtain all possible states of the transition we can use the following algorithm:

Given:

- a system $x_{t+1} \in \textbf{Post}(x_t, u_t)$
  where $x_t \in X$ is the state, $u_t \in U$ is control input, and $\textbf{Post}(x, u)$ is the set of all possible states from x under u.
- a set $X_{safe} \subseteq X$ of safe states

Find: the maximal controlled invariant set $X_{inv} \subseteq X_{safe}$

There are two equivalent approaches to computing X_{inv}:

- Approach I: performs set iteration and uses an operation called **CPre**.
- Approach II: is based on value iteration and only uses the **Post** operation.

In this study the **CPre** approach is used, and is further discussed. To perform set iteration using **CPre** on a set X_k,

$$X_0 = X_{safe}$$
$$X_{k+1} = X_k \cap \textbf{CPre}(X_k)$$

where $\textbf{CPre}(X_k) := x \in X \mid \exists u \in U : \textbf{Post}(x, u) \in X_k$
is the "controllable predecessor" of set X_k, i.e. the set of states that can be forced into X_k by the controller, regardless of the move of the environment. If the state set X is finite, the above set iteration is guaranteed to terminate in finite steps.

Computation of the MCIS requires three distinct computations:

1) System Loading: In this step, it computes $f(x, u)$, the set of all possible next states x^+ from x under u.
2) System Labeling: This iterates through all x of the transition and returns X_{safe}: the set of all states labelled as "safe."
3) Find Controlled Invariant Set: This function computes and returns the maximal controlled invariant set C and its associated controller K.

To illustrate the benefits of the parallelization, the System Loading function was parallelized. The original iterative function is outlined in Algorithm 1. As is shown, the algorithm first initializes the transition system in the form of distinct cells, each containing a large sparse matrix. The algorithm then iterates over all states, modifying each cell. Note that the loop is iterating over the states x_i and not over the number of cells.

In Algorithm 2 a slight modification is made to the data structure so that parallelization can take place. In MATLAB's parfor environment, different workers cannot modify the same element of a data structure. To prevent this from occurring, τ is redefined as a 3D matrix, slice the matrix along i_th layer. Each layer is, therefore, independent and can be processed by a different worker on the CPU. For even further optimization, only the tabular data is collected during the iteration step and the entire 3D matrix is constructed once the loop has completed.

Algorithm 1 Iterative Transition System Construction

```
Define transitions and number of nodes
Define other system specifications
tau = {}                                            ▷ Initialize cells
for k = 1:M do
$tau_k = sparse(N, N)$
$tau_k(N, N) = 1$
$tauend + 1 = tau_k$
end for
for i = 1:N-1 do                      ▷ Iterate over $x_i$ to create cells
$x_i = sys.gpart_x.idx2mid(i)$
$P0_{plus} = find(Map(p0, :))$
$P1_{plus} = find(Map(p1, :))$
$P2_{plus} = find(Map(p2, :))$
    for $p0_{plus} = P0_{plus}$ do
        for $p1_{plus} = P1_{plus}$ do
            for $p2_{plus} = P2_{plus}$ do
$k = sys.gpart_u.pnt2idx([p0_{plus}; p1_{plus}; p2_{plus}])$
Modify tau{k}(i,N) or tau{k}(i,po$_{plus}$)
            end for
        end for
    end for
end for
```

III. Feasibility Report

A feasibility test was run to evaluate the improvements of parallelization as well as to assess scalability. In this study, for proof-of-concept The System Loading step was parallelized across its iterator. Three scenarios were used: the original 7 node scenario, a 15 node scenario, and 23 node version representing Yellowstone National Park as a practical example as shown in figure 1. The computation time of the System Loading step for each scenario for straight iteration and parallelization were recorded and plotted. All tests were computed on a 36 core, 700 GB RAM cluster partition.

As can clearly be seen in figure 2, simple parallelization dramatically increases computational efficiency and reduces computational time for each scenario. Using simple iteration, while a simple 7-node scenario only took 71 minutes to compute, the real-world scenario took several days. It is likely to be even less scalable when fewer stringencies are placed on the environment setup However, when parallelized even the real-world scenario took less than two hours to compute. More computational resources will also be correlated with even more reduced computational time. While most user's local machines will not have this degree of resources, it does suggest that the computation of maximal controlled invariant sets using this method can be reasonably achieved if migrated to a cloud-based platform such AWS. MATLAB's Parallel Computing Toolbox already features support for AWS clusters. More information regarding creating cloud-based clusters for MATLAB can be found here.

Algorithm 2 Parallelized Transition System Construction

```
Define transitions and number of nodes
Define other system specifications
$[I, J, K, S] = deal(cell(N - 1, 1))$  ▷ Initialize tabular data
Iterate over all states to create cells
for (using parfor) i = 1:N-1 do ▷ Parallel Computation over $x_i$
$mytemp_{pk} = sparse(N, M)$
$x_i = sys.gpart_x.idx2mid(i)$
$P0_{plus} = find(Map(p0, :))$
$P1_{plus} = find(Map(p1, :))$
$P2_{plus} = find(Map(p2, :))$
    for $p0_{plus} = P0_{plus}$ do
        for $p1_{plus} = P1_{plus}$ do
            for $p2_{plus} = P2_{plus}$ do
$k = sys.gpart_u.pnt2idx([p0_{plus}; p1_{plus}; p2_{plus}])$
Modify $mytemp_{pk}$
            end for
        end for
    end for
$[Ii, Ji, Si] = find(mytemp_{pk})$
$Ki = repmat(i, size(Si))$
end for
$I = cell2mat(I(:))$
$J = cell2mat(J(:))$
$K = cell2mat(K(:))$
$S = cell2mat(S(:))$
$tau = ndSparse.build([I, J, K], S)$ ▷ Construct 3D Matrix
$tau(N, :, N) = 1$
```

In addition to purely static systems with high probability of transition success, this parallelized method as is will likely be successful in the following scenarios:

1) For simple, largely static systems with limited, isolated, known transience, one may still use the approach as is by breaking the transient system into multiple static systems. For example, a drawbridge connecting two nodes that can have both an open or a closed state can be represented as two separate systems, one closed and one open, and the user can simply choose the

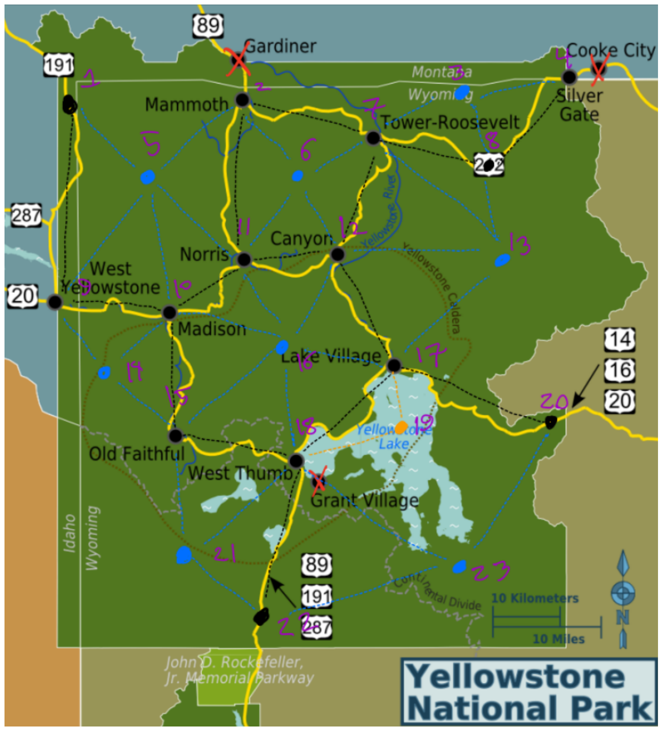

Fig. 1: Yellowstone National Park divided into 23 distinct nodes. Black nodes can be accessed by any vehicle whereas blue nodes can only be accessed by deployed vehicles. Node 19 contains an orange node that is intended to represent a space only occupiable by water-fairing robots. However, for simplicity this study treated this node as a blue node.

Nodes	Iterative	Parallelized
7	1.183	0.033
15	21.5	0.4
23	439	2

Fig. 2: Table demonstrating the number of hours needed to construct the transition system for each algorithm and number of nodes. Note that while all parallelized values are accurate, the iterative values are lower end approximations due to time limitations during data collection.

appropriate system depending on the bridge's current state.

2) High reliance on this method can be used in the case of nondeterministic mission target locations in unchanging environments with high transition success probability. An example of this is vehicle deployment for package pickup in clear weather, or where weather disruptions affect node accessibility in a predictable manner. In this case separate transition systems can be solved for different whether patterns.
3) In the case of slow transient environments this method can be used to evaluate an initial policy. If the transition probabilities of the current map are known, the policy can be evaluated for its chance of success. However, more research is needed to explore the viability of this option.

IV. Future Work

While this prototype shows promise for ensuring the safety of multi-robot systems, additional research is needed to determine if this method is practical in real world scenarios. This method assumes static transition systems with equal distance/energy consumption between nodes and guaranteed successful transition between states.

The above viable scenarios are of course quite limited. However, initial results do seem promising. Future work can thus be expanded into two categories: Computational optimization, and scenario expansion:

1) Parallelize the iteration over x_i in this CIS computation function for further improvements. In addition, further research in parallelizing the computation of CISs for finite transition systems may be benifitted by An efficient implementation of graph-based invariant set algorithm for constrained nonlinear dynamical systems
2) Computational Optimization: There are several approaches that are worth exploring for further computational optimization. These optimizations may act to save time and resources on currently compatible hardware or may also permit the computation of CISs on less powerful, and therefore more easily attainable, hardware.
    a) Compute a CIS that exists, but is not the maximum. Currently this code computes the maximum CIS (MCIS) of the transition system. However, in many cases, it is more computationally efficient to simply compute an existing CIS that is a subset of the total MCIS. If necessary, the MCIS can be computing as backup option if the initially computed CIS does not produce sufficient results.
    b) Evaluate if transition maps are chaotic (i.e. small changes in the transition system cannot be correlated with small changes in its corresponding MCIS). If they are not chaotic, then it is worth investigating if, if in the case where a CIS for an original map is known, if small changes in the transition system (e.g.. changes in nodes or number of vehicles) can be correlated to changes in the system's respective CIS. If so, starting with a "low-resolution" transition system and using an iterative method to expand to the larger, more defined system may be worth exploring. While more research is needed, two initial avenues may be worth exploring.
        i) Machine Learning - machine learning may be helpful both in the initialization of the system and the computation of its corresponding CIS by using the original transition system/CIS as a starting point and predicting the new initialization/CIS of the system. Using machine learning in this way may be reliable than

attempting to compute the system's initialization/CIS using pure learning. This method may also have value in scenario expansion as described in the next section.

ii) Policy Iteration - it may be worth exploring, that once the new transition system's initialization is calculated, the policy of the original system is used as a starting point to iterate through rather than computing the new policy from scratch.

c) Hierarchical computing: Often the total number of access points might be more than the computational resources allow for. In this scenario, it may be worth exploring the development of hierarchical maps. In this setup, rather than providing all nodes equal weight, we divide the nodes into a broad "low-resolution" map with fewer nodes, and assign the remaining nodes to be subsets of the low-res map nodes. In this way, a smaller transition system/CIS is computed for the low-res map and seperate each node subset can be treated as distinct transition systems with their own CISs. In many cases, this is a viable approach, since it is desired to have an overarching method to choose the general location of where robots should be directed, followed by a subroutine of what the robot should do once there. Breaking up the transition system has the potential to allow for the computation of CISs over vastly more complex transition systems.

d) Rewriting the current to a pre-compiled language such as C/C++. MATLAB is a scripting language, and for the compiled as it executes. Although great for development and prototyping, this choice may not be the best suited choice for fast computing. It is worth exploring optimization through simply using pre-compiled language which may significantly reduce computational overhead. C/C++ also uses OpenMP, a toolbox for parallelization.

3) Scenario Expansion:In many scenarios, the assumptions of a static environment, equal length transitions, and guaranteed transitions are too restrictive to produce meaningful results. There are several areas of research worth exploring and investigating as to their integration in the current code.

a) Expanding from a CIS to a Probabalistic CIS (PCIS): When success of transitions is not guaranteed, it may be worth exploring the development of CPIS. In this scenario, we assign each transition a probability and compute the CIS for a given probability threshold. More information can be found in Computing Controlled Invariant Sets.

b) Often, it is more valuable to have transition lengths represented as the amount of energy consumed during travel rather than simply using raw distance. However, it is often impractical to estimate transition energy manually. Deep learning approaches can be used to relate the topology of the landscape to expected energy consumption during travel Energy-based Legged Robots Terrain Traversability Modeling via Deep Inverse Reinforcement Learning. describes their usage of Maximum Entropy Deep Inverse Reinforcement Learning (MEDIRL) to predict the energy consumption of a given path and then determine the optimal path of travel. Once all transition lengths are computed, in many cases node location can be adjusted slightly to normalize each transition length.

c) Using machine learning for quickly computing the new CIS of a slightly modified transition system may also prove useful in computing the CIS of transient systems at a current time. For example, it may be impossible compute the CIS for all weather patterns of the transition systems ahead of time. However, it may be feasible to compute a CIS for clear weather/previous weather patterns and predict the change in CIS between previous weather pattern and the one currently impacting the environment. Similarly, if a natural disaster unpredictably affects the transition to/existence of a node on the map, using machine learning to modify the most recently known CIS to a current valid CIS may be a possibility for developing immediate intervention where time may be a premium.

d) Determination of system volatility: in some cases, transition systems can have frequent yet small changes. Sometimes these transitions have a period and we may be able to average the system and compute the averaged CIS. More information can be found here.

V. Additional Resources

Provided is a list of additional resources that may prove useful for further optimizing the computation of CISs in finite transition systems:

- The Computation of Full-complexity Polytopic Robust Control Invariant Sets
- Connected invariant sets for high-speed motion planning in partially-known environments
- Graph-based Computation of Control Invariant Sets: Algorithms, Analysis and Applications
- An efficient implementation of graph-based invariant set algorithm for constrained nonlinear dynamical systems

Acknowledgement

I would like to acknowledge the contributions of several others in the development of this report. Liren Yang started this work, and this report in large part is an extension of his work. the original MATLAB code for the 7-node system, the definition of the transition system used in this paper, and I had many conversations with him to acquire background knowledge for this report.

I would also like to thank Kwesi Rutledge for being a sounding board for many of the ideas presented in the Future Work section of this report.

Lastly, I would like to thank Glen Chou for proofreading this report and providing helpful edits.

References

[1] L. Yang, D. Rizzo, M. Castanier, and N. Ozay, "Parameter sensitivity analysis of controlled invariant sets via valueiteration."

[2] T. Anevlavis and P. Tabuada, "Computing controlled invariant sets in two moves," 2019.