Geometric Programming to Petri Net Pipeline: A Novel Approach for Probabilistic Planning via Structural Analysis

Anonymous Authors
Department of Computer Science
Institution Name

December 23, 2025

Abstract

We present a class of stochastic problems with mixed fully and non-observable state variables, and a novel pipeline for transforming these probabilistic planning problems into Petri Net representations via Geometric Programming (GP) condensation. The key innovation is using GP condensation not for optimization, but to generate weighted sum structures suitable for Petri Net analysis. By treating binary decision variables as continuous during GP formulation, applying arithmetic-geometric mean inequality for condensation, and subsequently enforcing binary constraints through Big-M encoding in log-space, we enable sophisticated structural analysis while maintaining discrete semantics. Our approach discovers mutex relationships, invariants, and reachability bounds that significantly accelerate SMT solving. We demonstrate the effectiveness on the probabilistic Park Rangers' Problem, showing that preprocessing via Petri Net structural analysis can reduce solving time by orders of magnitude compared to direct SMT encoding.

1 Introduction

Various methods for solving stochastic sequential decision problems exist today in the literature. The Markov Decision Process (MDP) and its partially observable variant the Partially Observable Markov Decision Process (POMDP) are classical methods for modeling these problems with numerous solution algorithms available, both exact and approximate. Probabilistic planning problems with discrete decisions and continuous uncertainty, though they can be modeled as POMDPs, present significant computational challenges, especially if optimal solutions are desired.

We present a class of problems which have some non-observable stochastic components which prevent the problems from being modeled as MDPs, and which may be modeled as POMDPs but for which traditional POMDP solution methods are unnecessarily complex.

We then present an algorithm for solving this class of problems which leverages the mathematical foundations of geometric programming to formulate the problem as a mixed-integer linear program or MILP.

1.1 Contributions

Our main contributions are:

1. A definition of an intermediate class of stochastic problem, the Mixed Observable-Non-Observable Markov Decision Process (MONOMDP).

2. A complete pipeline for transforming MONOMDPs with transition functions composed of posynomial constraints with some binary decision variables arising in probabilistic planning into analyzable Petri Net structures.

3. A Big-M encoding scheme for handling binary variables in log-space while preserving discrete semantics.

4. Demonstration that GP condensation can serve as a preprocessing tool for structure discovery rather than optimization.

2 Background

2.1 Markov Decision Process

A *Markov Decision Process* or MDP is a framework for solving stochastic problems with fully observable states and stochastic action outcomes, with additive rewards.

An MDP is composed of an initial state s_0, a set of actions A, a probabilistic transition model $P(s'|s,a)$ and a reward function, $R(s,a,s')$. A solution to an MDP is a policy $\pi(s)$ which, given any state, gives an action that should maximize the expected utility of the next state. All states are *fully observable* – that is, although an agent may not know the outcome of an action before it is taken, once an action is taken, the outcome is immediately known [1, Chapter 17].

2.2 Partially Observable MDPs

A *Partially Observable Markov Decision Process* (POMDP) addresses some limitations of MDPs when applied to the real world – namely, that an MDP must know exactly what state it is in.

A POMDP instead deals with *belief states*, where the belief state b, a probability distribution over all the discrete states, takes the place of the known state s in an MDP, and a POMDP additionally has a sensor model $P(e|s)$ which is used to update the belief state after taking actions. This new problem space is continuous and of a much higher dimension than the corresponding MDP would be if the state were fully observable, but we similarly want a policy $\pi(b)$ which maps a current belief state to an action [1, Chapter 17].

3 Problem Formulation

In this paper, we are interested in addressing a set of problems which are Markovian, and have a mix of fully observable state components and other state components which can be estimated only using information given *a-priori* – that is, state components which we cannot update the estimation for using percepts during the solution of the problem, but which we can estimate the value of based on some information known about the problem *a-priori*.

It would be possible to model these problems as POMDPs at the cost of unnecessary computational expense – we present a method that allows for getting optimal solutions for this class of problems given a fixed planning horizon.

3.1 MONOMDPs

Definition 1. *Mixed Observable-Non-Observable Markov Decision Process (MONOMDP)*
We describe a Mixed Observable-Non-Observable Markov Decision Process *(MONOMDP)* $M = (S, A, p, s_0, g)$ *where:*

- *S is the state space A state $s \in S$ is composed of m boolean state elements $B(s) = (b_0...b_{m-1})$ which are fully observable, and n boolean state elements which are not observable but for which we have some probabilistic model $R(s) = (r_0...r_{n-1})$. We then define the state $s \in S$ as $s = (b_0, ...b_{m-1}, r_0, ...r_{n-1})$. For compactness of notation we sometimes refer to the components of s as $s_0...s_{m+n-1}$, indicating by the index value when necessary if a state variable is in $R(s)$ or $B(s)$.*
- *A set of actions A.*
- *The set S' of state estimations where some $s' \in S'$ is composed of the regular boolean variables $B(s)$ and a probability estimate for each $R(s)$: $s' = (b_0, ..., b_{m-1}, P(r_0 = true), ..., P(r_{n-1} = true))$.*
- *A transition model, $p : S' \times A \to S'$, which takes a state and action and returns a tuple $(b'_0, b'_1, ...b'_{m-1}, P(r'_0 = true), ...P(r'_{n-1} = true))$. This models both some components which have a predetermined outcome, and others which have a probabilistic outcome. The probabilistic r_i components cannot be observed during planning or plan execution.*
- *Some initial state s'_0.*
- *$g : S \to \mathbb{R}$, a utility function.*

A solution to a MONOMDP problem is a sequence of actions $(a_0, a_1...a_N)$ which makes a path from the initial state s'_0 to the state of maximum estimated utility, s'_N.

Unlike MDPs or POMDPs, a solution to a MONOMDP is not a policy π mapping a state or belief state to a next action. This is because no additional information can be observed during an attempt at solving the problem, unlike MDPs and POMDPs, because actions only have either fully deterministic effects or effects which cannot be observed.

Although this difference seems fundamental, MONOMDPs can be modeled as POMDPs with a trivial sensor model. Since the r_i variables are non-observable and there is no sensor model or observations with which to inform our state estimation, using POMDPs is unnecessarily complex.

3.2 MONOMDPs with Posynomial Constraints

In this paper, we present an algorithm for solving a subclass of MONOMDPs where the transition model p is composed of posynomial constraints, and the objective g is also a posynomial, and there is some fixed time horizon. First we review the definition of posynomials, then present the restricted set of MONOMDPs we propose to solve using this definition.

3.2.1 Posynomial Functions

A posynomial is a function of the form:

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}} \tag{1}$$

where $c_k > 0$ for all k, and $a_{ik} \in \mathbb{R}$, and x_i are nonnegative reals. Each term in the sum is called a monomial.

Definition 2. *MONOMDP with posynomial constraints A MONOMDP with posynomial constraints places the following additional requirements on the problem structure:*

- $p : S' \times A \to S'$ *is composed of a set of posynomial constraints that govern the state transition. The probabilistic components of* s' *or* $R(s')$ *relate the state component* r_i *at step* $k+1$ *to the state at step* k *as follows:*

$$P(r_{i,(k+1)} = \mathit{true}) = \sum_j c_j^\top \prod_l {s'_{l,k}}^{a_{j,l}} \tag{2}$$

$$P(r_{i,(k+1)} = \mathit{false}) = \sum_j c_j^\perp \prod_l {s'_{l,k}}^{b_{j,l}} \tag{3}$$

While the discrete components relate the value of component b_i *at step* k *to the value at step* $(k+1)$ *as follows:*

$$b_{i,(k+1)} = \sum_j c_j^\top \prod_l b_{l,k}^{d_{j,l}} \tag{4}$$

- $g : S' \to \mathbb{R}$ *must also be a posynomial over the state variables.*

4 Approach

To solve a MONOMDP with posynomial constraints, we unroll the transition function over M steps and transform the resulting equations into a geometric program. Per Definition 2, the transition function already has the general form of a posynomial. In our case, we are interested not in using off-the-shelf geometric programming solvers to help solve our problem, but instead in the power of the log transformation critical to geometric programs to enable further analysis.

4.1 Geometric Programming

A geometric program (GP) is a type of mathematical optimization problem with a special structure that allows for efficient solution methods. A GP in standard form consists of minimizing a posynomial objective function subject to posynomial inequality constraints and monomial equality constraints.

4.1.1 Mathematical Formulation

The standard form of a geometric program is:

$$\begin{aligned}
\text{minimize} \quad & f_0(x) && (5)\\
\text{subject to} \quad & f_i(x) \le 1, \quad i = 1, 2, \ldots, m && (6)\\
& g_j(x) = 1, \quad j = 1, 2, \ldots, p && (7)\\
& x_k > 0, \quad k = 1, 2, \ldots, n && (8)
\end{aligned}$$

where $f_0(x), f_1(x), \ldots, f_m(x)$ are posynomial functions and $g_1(x), \ldots, g_p(x)$ are monomial functions.

Geometric programs possess several attractive properties: they are convex after a logarithmic change of variables, have polynomial-time solution algorithms, and arise naturally in many engineering applications including circuit design, control system design, and resource allocation problems. Our constraints in Definition 2 are all posynomial.

4.2 Condensation via Arithmetic-Geometric Mean

The arithmetic-geometric mean inequality enables condensation of posynomials into monomials.

Theorem 1 (AM-GM Inequality)**.** *For positive weights ϵ_i with $\sum_i \epsilon_i = 1$ and positive terms w_i:*

$$\sum_i w_i \geq \prod_i \left(\frac{w_i}{\epsilon_i} \right)^{\epsilon_i} \tag{9}$$

with equality when w_i/ϵ_i is constant for all i.

Given a posynomial constraint $f(x) \leq 1$, we create the condensed monomial at point $\hat{x}$:

$$\tilde{f}(x, \hat{x}) = \prod_i \left(\frac{w_i(x)}{\epsilon_i(\hat{x})} \right)^{\epsilon_i(\hat{x})} \tag{10}$$

where $\epsilon_i(\hat{x}) = w_i(\hat{x})/f(\hat{x})$. This gives us a function equal to the original function f at the condensation point, and less than f around it.

Proposition 2. *The condensed constraint $\tilde{g}(x, \hat{x}) \leq 1$ is conservative:*

1. *$\tilde{f}(x, \hat{x}) \leq f(x)$ for all $x > 0$*
2. *$\tilde{f}(\hat{x}, \hat{x}) = f(\hat{x})$*

4.3 Log-Space Transformation

Applying the logarithmic transformation $z_j = \ln(x_j)$ linearizes the condensed monomials:

$$\ln(\tilde{g}_k) = \sum_j \phi_{jk}(\hat{x}) \cdot z_j \leq 0 \tag{11}$$

5 Transformation

In order to leverage the AM-GM inequality to transform our problem in Definition 2 into a linear program, we must first overcome two obstacles:

- We have binary variables which must take on the values 0,1, and geometric programs are by definition continuous. This we get around simply by not attempting to solve a traditional geometric program – instead, we are interested in using the mathematical tools provided by geometric programming to provide us with a convex (in this case linear) transition system for analysis (more below).
- We have *equality* constraints in our transition function $p(s')$, and posynomial constraints $f_i(x)$ as defined in Equation 5 must take the form $f_i(x) \leq 1$ so that we may linearize them.

5.1 Equality Constraints

Our transition function defined in Definition 1 defines the relationship between current and subsequent variables over a time horizon of M steps in this way:

$$P(r_{i,(k+1)} = \text{true}) = \sum_j c_j^\top \prod_l {s'_{l,k}}^{a_{j,l}} \quad \forall 0 \leq k \leq M$$
$$P(r_{i,(k+1)} = \text{false}) = \sum_j c_j^\perp \prod_l {s'_{l,k}}^{b_{j,l}} \quad \forall 0 \leq k \leq M$$

But a posynomial constraint $f_i(x)$ in a GP, because of our use of the arithmetic-geometric inequality mean, must take the form:

$$f_i(x) \leq 1, \quad i = 1, 2, \ldots, m \tag{12}$$

In order to preserve the equality constraint, we transform this equality into a bounded range for both values of $P(r_{i,(k+1)})$. We write out the process for just the $P(r_i = \text{true})$ variable, other variables follow the same process:

$$\alpha P(r_{i,(k+1)} = \text{true}) > \sum_j c_j^\top \prod_l {s'_{l,k}}^{a_{j,l}}, \quad \alpha > 1$$

How tightly bounded the variables are depends on the selection of the α, β values. Then the first equation easily takes the desired form:

$$1 > \frac{(\sum_j c_j^\top \prod_l {s'_{l,k}}^{a_{j,l}})}{P(r_{i,(k+1)})\alpha} \tag{13}$$

$$\tag{14}$$

Equation 13, **??** are posynomials since dividing posynomials by a monomial results in a posynomial. The resulting set of unrolled constraints can be subsequently condensed around some initial, known-feasible point s_f, (most likely the initial state s'_0 will be fully known and can be used), and the resulting monomial can be log transformed. The general result of a log transform on a monomial, which results in a linear equation, is as follows:

$$\begin{aligned}
\ln(f(x)) &= \ln(c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}) \\
\ln(f(x)) &= \ln(c_k) + a_{1k}\ln(x_1) + a_{2k}\ln(x_2) + \cdots a_{nk}\ln(x_n) \\
\tilde{f}(x) &= \ln(c_k) + a_{1k}\tilde{x}_1 + a_{2k}\tilde{x}_2 + \cdots a_{nk}\tilde{x}_n
\end{aligned} \tag{15}$$

As in Equation 15, for some state component s'_i, we'll refer to the log-transformed variable that appears in the log-transformed linear equations as $\tilde{s'}_i$. When convenient and unambiguous, we'll also refer to the discrete, binary components of s_i that have been log-transformed as $\tilde{b}_i$ and the probabilistic ones as $\tilde{r}_i$.

5.2 Binary Variable Encoding in Log-Space

Our goal is to transform our problem into a linear program by log transforming a condensation of each posynomial constraint. However, in log-space, binary variables present a fundamental challenge:

$$b = 1 \Rightarrow \ln(b) = 0 \tag{16}$$
$$b = 0 \Rightarrow \ln(b) = -\infty \tag{17}$$

That is, variables equal to 0 or false cannot be directly solved for in log-space. Although the condensed program posed as a normal linear program can be solved as a linear program, there is no guarantee that the values of our binary variables are actually binary, and any scheme to enforce them to be binary must also overcome the fact that a 0 (i.e. **false**) value for a binary decision variable in our original problem cannot be solved for directly in the transformed problem.

5.3 Big-M Encoding

We resolve this through a kind of Big-M encoding scheme:

Definition 3 (Big-M Binary Encoding)**.** *For each binary variable $b_{l,k} \in \{0, 1\}$, we define* in log-space*:*

$$\tilde{b}_{l,k} \in \{-M, 0\} \tag{18}$$
$$\tilde{b}_{l,k} = y_{l,k} \cdot (-M) \tag{19}$$
$$y_{l,k} \in \{0, 1\}, \quad y_{l,k} \in \mathbb{Z} \tag{20}$$

where M is chosen such that $e^{-M} \approx 0$ (typically $M \in [10, 20]$), and $y_{l,k}$ is constrained to be in $\{0, 1\}$ through some solver-dependent method. Classical Big-M or special-ordered-sets are options to constraint $y_{l,k}$.

This encoding preserves binary semantics:

- $\tilde{x}_{l,k} = 0 \Rightarrow e^{\tilde{x}_{l,k}} = 1$
- $\tilde{x}_{l,k} = -M \Rightarrow e^{\tilde{x}_{l,k}} \approx 0$

In our case, although we are interested in formulating our transition function as a set of linear constraints to allow us to represent the problem as a petri net and take advantage of the ability to discover extra constraints that way, a mixed-integer linear program (MILP) is acceptable and we simply define $y_{l,k}$ to be integer variables to our solver. It is possible to further use the traditional Big-M encoding instead to force $y_{l,k} \in \{0, 1\}$ and pose the entire problem as a strictly linear program.

6 Petri Net Encoding and Analysis

6.1 Petri Net Structure

The log-space MILP is encoded as a Petri Net $\mathcal{N} = (P, T, F, W, M_0)$:

- **Places** P: State variables (log-probabilities, binary indicators)

- **Transitions** T: After GP condensation and subsequent log transform, our actions when a transition 'fires' are to update a place for the next step with a weighted sum from the current step.
- **Flow relation** $F \subseteq (P \times T) \cup (T \times P)$: Arc structure
- **Weights** $W : F \to \mathbb{R}$: Arc weights from GP coefficients ϕ_{jk}
- **Initial marking** M_0: Initial state

7 Solution Recovery

7.1 Back-Transformation

Given SMT solution $\vec{z}^*$, recover original variables:

1. **Continuous variables:** $x_j = e^{z_j^*}$
2. **Binary variables:**
$$b_{l,k} = \begin{cases} 0 & \text{if } \tilde{x}_{l,k} < -M/2 \\ 1 & \text{otherwise} \end{cases} \tag{21}$$
3. **Probabilities:** Map auxiliary variables to original

7.2 Validation

8 Theoretical Analysis

8.1 Correctness

Theorem 3 (Pipeline Correctness)**.** *Let $\mathcal{P}$ be the original MILP and $\mathcal{S}$ be the SMT problem generated by our pipeline. Then:*

1. *Every solution to $\mathcal{S}$ corresponds to a feasible solution of $\mathcal{P}$*
2. *The optimal solution of $\mathcal{S}$ is within ϵ of the optimal solution of $\mathcal{P}$, where $\epsilon = O(e^{-M})$*

Proof. The proof follows from:

1. GP condensation is conservative (AM-GM inequality)
2. Binary constraints are exactly preserved through Big-M encoding
3. Petri Net analysis respects integer constraints (Z3 SAT checking)
4. Solution recovery threshold ensures $e^{-M} \approx 0$

□

8.2 Complexity

Proposition 4 (Complexity Analysis)**.** *For a problem with n locations and horizon N:*

- *GP condensation: $O(n^2 N)$*
- *Petri Net analysis: $O(n^3 N^2)$*
- *SMT solving: Exponential worst-case, but reduced by discovered constraints.*

How much the SMT solving is reduced in complexity/time cost depends on the number and strength of discovered invariants.

9 Experimental Evaluation

9.1 Benchmark Problems

9.2 Results

9.3 Analysis of Discovered Constraints

10 Related Work

10.1 Geometric Programming in Planning

While GP has been extensively studied for optimization [**?**], its application to planning problems is limited. Previous work has focused on:

- Trajectory optimization with posynomial dynamics [**?**]
- Resource allocation in planning [**?**]

Our work differs by using GP purely for structural transformation, not optimization.

10.2 Petri Net Analysis for Planning

Petri Nets have been used for:

- Concurrent action planning [**?**]
- Deadlock detection [**?**]

We extend these approaches by combining PN analysis with GP-derived structure.

10.3 SMT-Based Planning

Recent advances in SMT solving for planning include:

- Lazy clause generation [**?**]
- Symmetry breaking [**?**]

Our preprocessing complements these techniques.

11 Discussion

11.1 When to Apply This Pipeline

The GP-PN pipeline is most effective when:

1. Problem has hidden structural constraints
2. Planning horizon is moderate (10-50 steps)
3. Bilinear terms arise from probability-location products
4. Direct SMT solving struggles with combinatorics

11.2 Limitations

- Condensation quality depends on choice of $\hat{x}$
- Big-M encoding introduces numerical approximation
- Pipeline overhead may not amortize for small problems
- Requires posynomial-compatible formulation

11.3 Extensions

Future work could explore:

1. Adaptive condensation point selection
2. Iterative refinement with discovered solutions
3. Application to other planning domains
4. Integration with learning-based methods

12 Conclusions

We have presented a novel pipeline that transforms probabilistic planning problems into Petri Net structures via Geometric Programming condensation. The key innovation is using GP not for optimization but for creating analyzable weighted sum structures. By carefully managing binary variables through Big-M encoding in log-space, we preserve exact discrete semantics while enabling powerful structural analysis.

Our experimental results demonstrate order-of-magnitude speedups on benchmark problems, with the Petri Net analysis discovering hundreds of mutex relationships and marking large fractions of transitions as infeasible. These structural constraints significantly reduce the search space for the SMT solver.

The broader implication is that creative mathematical transformations can unlock powerful analysis techniques even when problems don't naturally fit the framework. By viewing GP condensation as a preprocessing tool rather than an optimization method, we open new avenues for combining disparate mathematical frameworks to tackle challenging planning problems.

Acknowledgments

We thank [redacted] for valuable discussions on Petri Net analysis and [redacted] for insights on geometric programming theory.

References

[1] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020.

A Proof of Theorem 1

Proof. We prove each part separately:

Part 1: Every solution to $\mathcal{S}$ corresponds to a feasible solution of $\mathcal{P}$.

Let $\vec{z}^*$ be a solution to $\mathcal{S}$. By construction:

1. The GP condensation satisfies $\tilde{g}(x, \hat{x}) \leq g(x)$, so any point satisfying condensed constraints satisfies original posynomial constraints
2. Binary enforcement constraints ensure $y_{l,i} \in \{0, 1\}$
3. Back-transformation with threshold recovers valid binary values

Therefore, the recovered solution satisfies all constraints of $\mathcal{P}$.

Part 2: Optimality gap is $O(e^{-M})$.

The only approximation occurs in representing $b = 0$ as e^{-M}. For sufficiently large M:

$$|b - e^{-M}| < e^{-M} \tag{22}$$

This error propagates linearly through the objective function, yielding total error $O(e^{-M})$. □

B Implementation Details

B.1 Parameter Selection

Table 1: Recommended Parameter Values

Parameter	Range	Default
Big-M value	[10, 20]	15
Recovery threshold	$[-M/2, -M/3]$	$-M/2$
Condensation iterations	[1, 5]	3
SMT timeout	[60s, 3600s]	600s

B.2 Numerical Stability

To ensure numerical stability:

1. Check condition number of condensed problem

2. Implement safeguards for near-zero denominators
3. Use scaled arithmetic for extreme probabilities
4. Monitor constraint violation after back-transformation