Constraint Partitioning in Penalty Formulations for Solving Temporal Planning Problems*

Benjamin W. Wah** and Yixin Chen***

Abstract

In this paper, we study the partitioning of constraints in temporal planning problems formulated as mixed-integer nonlinear programming (MINLP) problems. Constraint partitioning is attractive because it leads to much easier subproblems, where each is a significant relaxation of the original problem. Moreover, each subproblem is very similar to the original problem and can be solved by any existing solver with little or no modification. Constraint partitioning, however, introduces global constraints that may be violated when subproblems are evaluated independently. To reduce the overhead in resolving such global constraints, we develop in this paper new conditions and algorithms for limiting the search space to be backtracked in each subproblem. Using a penalty formulation of a MINLP where the constraint functions of the MINLP are transformed into non-negative functions, we present a necessary and sufficient extended saddle-point condition (ESPC) for constrained local minimization. When the penalties are larger than some thresholds, our theory shows a one-to-one correspondence between a constrained local minimum of the MINLP and an extended saddle point of the penalty function. Hence, one way to find a constrained local minimum is to increase gradually the penalties of those violated constraints and to look for a local minimum of the penalty function using any existing algorithm until a solution to the constrained model is found. Next, we extend the ESPC to constraint-partitioned MINLPs and propose a partition-and-resolve strategy for resolving violated global constraints across subproblems. Using the discrete-space ASPEN and the mixed-space MIPS planners to solve subproblems, we show significant improvements on some planning benchmarks, both in terms of the quality of the plans generated and the execution times to find them.

Keywords and phrases: Constraint partitioning, extended saddle-point condition, penalty function, local search, mixed space planning, nonlinear constraints, temporal planning.

1 Introduction

A temporal planning problem involves arranging actions and assigning resources in order to accomplish given tasks and objectives over a period of time. It can be defined by a state space with discrete, continuous, or mixed variables; a discrete or continuous time horizon; a set of actions defining valid state transitions; a set of effects associated with each action; a set of constraints to be satisfied in each state or throughout an action; and a set of goals to be achieved.

*Research supported by National Science Foundation Grant IIS 03-12084 and National Aeronautics and Space Administration Grant NCC 2-1230.

** Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, Urbana, IL 61801, http://manip.crhc.uiuc.edu, wah@uiuc.edu.

*** Department of Computer Science, Washington University, St. Louis, MO 63130.
In this paper, we formulate a planning problem as a mixed-integer nonlinear programming (MINLP) problem. Such a formulation allows us to develop a formal mathematical foundation when partitioning a large planning problem by its constraints into subproblems (stages). The MINLP formulation of the problem when partitioned into \( N + 1 \) subproblems is as follows:

\[
(P_t) : \quad \min_z J(z) \\
\text{subject to } \quad h^{(t)}(z(t)) = 0, \quad g^{(t)}(z(t)) \leq 0, \quad t = 0, \ldots, N, \quad \text{(local constraints)}, \\
\text{and } \quad H(z) = 0, \quad G(z) \leq 0, \quad \text{(global constraints)}. 
\]

Here, Stage \( t, t = 0, \ldots, N, \) has local state vector \( z(t) = (z_1(t), \ldots, z_m(t))^T \) of \( u_t \) mixed variables; \( h^{(t)} = (h_1^{(t)}, \ldots, h_m^{(t)})^T \) is a vector of \( m_t \) local equality-constraint functions that involve \( z(t) \); \( g^{(t)} = (g_1^{(t)}, \ldots, g_r^{(t)})^T \) is a vector of \( r_t \) local inequality-constraint functions of \( z(t) \); \( H = (H_1, \ldots, H_p)^T \) is a vector of \( p \) global equality-constraint functions that involve \( z = \bigcup_{i=0}^N z(i) \); and \( G = (G_1, \ldots, G_q)^T \) is a vector of \( q \) global inequality-constraint functions of \( z \). Note that \( z(t) \) includes all variables that appear in one or more of the local constraints in Stage \( t \), and that \( z(0), \ldots, z(N) \) may overlap with each other since the partitioning is by constraints.

We assume that \( J \) is continuous and differentiable with respect to its continuous variables and is lower bounded. Further, \( g \) and \( h \) can be unbounded, discontinuous, non-differentiable, and not in closed form. These assumptions are reasonable for AI planning problems, whose constraint functions may be discontinuous and not in closed form and whose objective functions are continuous and differentiable in the continuous subspace. A solution to \( P_t \) is a plan that involves an assignment of \( z \) and that satisfies all the constraints.

To illustrate the constrained formulation of a planning problem, consider the toy problem in Figure 1 solved by ASPEN [9]. The problem involves scheduling four activities: \( \text{act.}_1 \) and \( \text{act.}_2 \) of type A1 and \( \text{act.}_3 \) and \( \text{act.}_4 \) of type A2, over a discrete horizon of 60 seconds. Its goal is to satisfy the nineteen constraints, \( E_1 \) through \( E_{19} \), on positive and negative facts and preconditions and effects of actions, while minimizing the total power_resource used. Among the 19 constraints in the initial schedule in Figure 1, \( E_1, E_2, E_3, E_4, E_6, \) and \( E_{15} \) are not satisfied.

In a MINLP formulation of the toy example, each of the nineteen constraints in Figure 1 is transformed into one or more equivalent constraints. We use two variables \( s(a) \) and \( e(a) \) to denote, respectively, the starting and ending times of activity \( a \). For each state, we assign a vector of state variables to denote their values indexed by time. For example, we use \( c(t) \) to denote the color_state at time \( t \), which can be set to 0 (red), 1 (blue), or 2 (green) \( (c(t) = 2 \) means that the color_state at time \( t \) is green); \( p(t) \) to denote the power_supply at \( t \); and \( w(t) \) to denote the power_usage at \( t \). The following illustrates a small portion of the resulting constraints encoded:

\[
\begin{align*}
(c1) \quad w(t) & \leq p(t) \leq 25, \quad \forall t = 0, \ldots, 60; \quad \text{\( \text{power}_{\text{resource}} \) capacity constraint} \\
(c2) \quad 0 & \leq s(\text{act.}_3) - e(\text{act.}_1) \leq 30; \quad \text{\( \text{act.}_1 \) ends before start of \( \text{act.}_3 \) by [0,30]} \\
(c3) \quad s(\text{act.}_1) = t & \implies c(t) = 2; \quad \forall t = 0, \ldots, 60; \quad \text{\( \text{color}_{\text{state}} \) constraint for \( \text{act.}_1 \)} \\
(c4) \quad s(cc_b) = t & \implies c(t) = 1; \quad \forall t = 0, \ldots, 60; \quad \text{\( \text{color}_{\text{changer}} \) \( cc_b \) effect constraint}
\end{align*}
\]

The constraints are either equality or inequality constraints (such as \( c1 \) and \( c2 \)), or deduction constraints (such as \( c3 \) and \( c4 \)). A deduction constraint \( A \implies B \), where \( A \) and \( B \) are equality or inequality constraints, can be encoded as an equivalent equality constraint \( H(A \implies B) = 0 \):

\[
H(A \implies B) = \begin{cases} 
0 & \text{if } A \text{ is false, or } A \text{ and } B \text{ are both true} \\
\text{numerical violation of } B & \text{if } A \text{ is true but } B \text{ is false.}
\end{cases}
\]

For example, the equivalent equality constraint encoding \( c3 \) returns 0 if \( s(\text{act.}_1) = t \) is false; otherwise, it returns the value of \( (c(t) - 2) \).

Artificial Intelligence (accepted to appear)
model toy HORIZON_START = 1998-1/00:00:00; horizon_duration = 60s; time_scale = second;
parameter_variable color_sv domain = ("red", "blue", "green");
state_variable color sv states = ("red", "blue", "green");
default state = "red";
resource power type = non_depletable; capacity = 25; min_value = 0;
activity color_changer color c; duration = 1; reservations = color_sv change_to c;
activity A1 duration = [10,20]; constraints = ends_after start of A2 by [0,30];
reservations = power use 10, color_sv must be "green";
activity A2 duration = 10; reservations = power use 20, color_sv must be "blue";
prefer linearly less resource power total value.; /optimization objective
// initial infeasible schedule
A1 act.1 start_time = 0; duration = 15;; A1 act.2 start_time = 20; duration = 10;;
A2 act.3 start_time = 30; duration = 10;; A2 act.4 start_time = 50; duration = 10;;

A general approach for solving a large constrained optimization problem is to select iteratively
a set of its variables to which values can be assigned according to the structural characteristics of
the domain, and to partition the problem into subspaces by setting the variables selected to specific
values. Systematic-search methods may set the values of variables in some predefined order or in an
order independent of the interactions among variables. By considering variable interactions, intelli-
gen backtracking employs variable/value ordering to order the subproblems generated, pre-filters
partial inconsistent assignments to eliminate infeasible subproblems, and prunes subproblems with
inferior bounds computed by relaxation or approximation. Alternatively, iterative-repair methods
operate on the full dimensionality of the starting problem and consider the interaction of each as-
ignment with all its variables. In general, it is difficult to make full use of the interactions among
variables and subproblems in the selection and assignment of variables.

In this paper, we propose a new approach called constraint partitioning that decomposes the
constraints of a problem into a conjunction (\land) of subproblems, each with local constraints and
related to others by global constraints (Figure 2). Constraints that relate to only variables in
one subproblem are local constraints, whereas constraints that relate to local variables as well as
shared entities and variables across subproblems are global constraints. Since shared entities
across subproblems must be consistent, additional global constraints may be added to enforce

Figure 1: A toy example from ASPEN [9] whose goal is to find a valid schedule that completes 4 activities,
act.1 and act.2 that are instances of type A1 and act.3 and act.4 that are instances of type A2, while
minimizing the total power_resource used. Based on the initial infeasible schedule, the 19 constraints are
partitioned into 3 stages, \{E_1, E_5, E_9, E_{11}\}, \{E_2, E_3, E_6, E_7, E_{10}, E_{12}, E_{13}, E_{15}\}, and \{E_4, E_8, E_{14}\}, and 4
global constraints \{E_{16}, E_{17}, E_{18}, E_{19}\}. A local constraint remains associated with a stage even when activities
are rescheduled. The number of iterations to solve the problem is reduced from 16 taken by ASPEN to
12 after partitioning.
Figure 2: An illustration of constraint partitioning that decomposes $P$ into a conjunction ($\wedge$) of three subproblems and a set of global constraints to be resolved, where the complexity of each subproblem is substantially smaller than that of $P$. The set of global constraints $G$ includes constraints in $P$ that span across variables in multiple subproblems and new constraints added to maintain the consistency of shared entities and variables across subproblems.

Figure 3: Two extreme configurations when partitioning the constraints of a problem. A constraint-partitioned problem can be solved by first evaluating the subproblems independently, using possibly existing methods and disregarding some or all of the global constraints, and by resolving the violated global constraints through systematic backtracking of subproblem evaluations. The advantage of evaluating subproblems independently is that each is much more relaxed than the original problem and requires significantly less time to solve. The difficulty, however, lies in the resolution of violated global constraints because they are defined in an exponentially large space across the subproblems. Even though the subproblems may be organized into stages, dynamic programming and the Principle of Optimality [3] cannot be applied because a partial feasible plan that dominates another partial feasible plan in one stage will fail to hold when the dominating plan violates a global constraint in a later stage. Without dominance, resolving a violated global constraint may invalidate the solutions of subproblems found already and require backtracking of their evaluations. The complexity due to backtracking is hard to characterize precisely because it depends on the aggregate search space across all the stages in which global constraints can be satisfied. In the following, we illustrate this complexity in two extreme cases.

Figure 3 illustrates two extreme configurations when partitioning the constraints of a problem: one whose stages have totally overlapped variable sets and the other with totally disjoint variable sets.

 Artificial Intelligence (accepted to appear)
sets. In each stage, the outermost box denotes the search space of the stage; the first inner box denotes all feasible solutions that satisfy the local constraints in that stage; and the innermost shaded box denotes points that satisfy our proposed ESPC (which also satisfy the local constraints). In this paper, we show that ESPC is necessary and sufficient for all constrained local minima, which means that only those points that satisfy ESPC in each stage should be considered in resolving violated global constraints. Let \( s_i \) be the set of points that satisfy ESPC in stage \( i, i = 0, \ldots, N \).

In Figure 3a, when the stages have totally overlapped variable sets, each constraint in the original problem can be assigned to exactly one stage. Because every variable is shared across all the stages, new global constraints must be introduced to maintain its consistency across the subproblems. These global constraints are defined in a search space whose worst-case complexity is bounded by \( |s_0 \cap s_1 \cap \cdots \cap s_N| \). Since the number of such global constraints as well as \( s_i, i = 1, \ldots, N \), can be large, the complexity for resolving the global constraints can be very high. In contrast, in Figure 3b, when the stages have totally disjoint variable sets, it is likely that most constraints cannot be assigned as local constraints and will remain as global. These global constraints will need to be resolved in a search space whose worst-case complexity is bounded by \( |s_0| \cdot |s_1| \cdots |s_N| \). Due to the large number of such global constraints, the overhead for resolving them will likely to be high as well. In short, the minimization of the overhead in resolving violated global constraints entails trade-offs among the number of shared entities and variables across the subproblems, the number of global constraints involved, and the search space where the global constraints are defined.

In this paper, we analyze the constraints of temporal planning problems in order to partition them into a small number of simpler subproblems (stages). In general, it is hard to develop a good partitioning algorithm that minimizes the time to solve a planning problem because the relation between the time to solve a subproblem and that to resolve violated global constraints is complex and unknown. In this paper, we exploit the temporal locality of constraints in planning problems when partitioning them into stages. Starting from an initial (possibly infeasible) schedule, we partition the constraints along the horizon into a small number of stages, each with an approximately equal number of constraints. For example, Figure 1 (resp. Figure 4) shows the nineteen (resp. 3,687) constraints of an initial infeasible schedule generated by ASPEN [9] in solving the toy example (resp. CX1-PREF with sixteen orbits). After partitioning the constraints into three (resp. four) stages, the resulting problem has fifteen (resp. 3,580) local constraints and four (resp. 107) global constraints. Since some violated global constraints may become satisfied or new constraints corresponding to new actions may be added as planning progresses, we also study algorithms to determine a suitable number of stages and to repartition the constraints periodically in order to balance the number of violated constraints in each stage.

Our major goal in this paper is to develop the theory and the corresponding algorithms for

Artificial Intelligence (accepted to appear)
resolving violated global constraints when temporal planning problems are partitioned by their constraints into stages. Specifically, there are three contributions in this paper.

a) We show in Section 3 the necessary and sufficient extended saddle-point condition (ESPC) that governs all constrained local minima, when a MINLP problem is formulated in a penalty function with non-negative (transformed) constraint functions. This paper is the first to show that each constrained local minimum of the MINLP has a one-to-one correspondence to an extended saddle point of the penalty function when its penalties are sufficiently large. Using this result, one way to look for a constrained local minimum of the MINLP is to increase gradually the penalties of violated constraints in the penalty function and to search repeatedly local minima of the penalty function by an existing algorithm until a feasible solution to the constrained model is found.

b) We present in Section 4 that the ESPC can be decomposed for constraint-partitioned MINLPs. Each decomposed ESPC is defined with respect to a subproblem consisting of its local constraints and an objective function that is made up of the objective of the original problem and biased by a weighted sum of the violated global constraints. As such, each subproblem is very similar to the original problem and can be solved by the same planner with little or no modification.

c) We describe a partition-and-resolve procedure in Section 4.2. The procedure iterates between calling a planner to solve the constraint-partitioned subproblems, and using a constraint-reweighting strategy to resolve the violated global constraints across the subproblems. In Section 5, we demonstrate significant improvements in using the discrete-space ASPEN and the mixed-space MIPS as basic planners to solve some large-scale benchmarks. For example, the problem in Figure 1 (resp. 4) can be solved by ASPEN in 16 (resp. 12,043) iterations and by our implementation in 12 (resp. 1,102) iterations with the same (resp. better) quality.

## 2 Previous Work

In this section, we summarize some existing work related to AI planning and nonlinear optimization. Our survey shows that existing approaches solve a problem directly while taking all its constraints into consideration.

### 2.1 Existing Temporal Planning Methods

Figure 5 classifies existing AI planning and scheduling methods based on their state and temporal representations and the search techniques used.

**Discrete-time discrete-state methods** consist of systematic searches, heuristic searches, local searches, and transformation methods. Systematic searches that explore the entire state space

---

**Figure 5:** A classification of existing planning and scheduling approaches.
2.2 Existing Mathematical Programming Methods

Existing Mathematical Programming Methods

are complete solvers. After decomposing a search space into subspaces, they evaluate each as a complete planning problem. Examples include the generic A* algorithm, UCPOP [32], Graphplan [5], STAN [28], PropPLAN [13], and System R [27].

Heuristic solvers explore the search space by a tree search guided by heuristics in order to estimate the distance from a state to the goal state. They do not have means to resolve violated global constraints when the original planning problem is partitioned by its constraints into subproblems. They are not guaranteed to find feasible plans because their success depends on the guidance heuristics used. Examples include HSP [6], FF [18], AltAlt [31], GRT [37] (and its extension to MO-GRT [38]), and ASPEN [9]. Last, transformation methods convert a problem into a constrained optimization or satisfaction problem, before solving it by an existing solver. Examples in this class include SATPLAN [22] Blackbox [23], and ILP-PLAN [24].

Discrete-time mixed-state methods consist of systematic searches, heuristic searches, and transformation methods. Similar to discrete-time discrete-state methods, methods in this class do not partition the constraints of a planning problem. Examples include SIPE-2 [52], O-Plan2 [46], Metric-FF [18], GRT-R [37], and LPSAT [54].

Continuous-time mixed-state methods can be classified into systematic, heuristic, and local searches. Again, constraints are not partitioned in these methods. Examples include LPG [16], MIPS [12], Sapa [45], ZENO [33], SHOP2 [30], TALplanner [10], and Europa [21].

In summary, existing planners solve a problem as a whole without partitioning its constraints, or transform it into another form before solving it by existing methods. In this paper, we propose to augment existing approaches by constraint partitioning and decompose the constraints of a large problem into subproblems of a similar form before solving them by existing planners. Instead of developing a new planner based on ESPC to solve the small subproblems, using an existing planner is more effective because it performs well in solving small problems, besides saving a lot of development efforts. We demonstrate our approach in Section 5 after formulating the objectives and the constraints of the subproblems solved by ASPEN and MIPS.

2.2 Existing Mathematical Programming Methods

In this section, we survey existing methods on continuous and mixed-integer optimization. Although many of these methods cannot be applied to solve planning problems because they have requirements, such as continuity, differentiability, and convexity, that are not satisfied in planning problems, it is necessary to understand their limitations. The concepts of saddle points and penalty formulations are important and form the basis of our theory presented in Section 3.

Continuous nonlinear programming (CNLP) methods. Consider the following CNLP:

\[
(P_c) : \quad \min_x f(x) \quad \text{where} \quad x = (x_1, \ldots, x_v)^T \in \mathbb{R}^v \\
\text{subject to} \quad h(x) = (h_1(x), \ldots, h_m(x))^T = 0 \quad \text{and} \quad g(x) = (g_1(x), \ldots, g_r(x))^T \leq 0,
\]

where \( f \) is continuous and differentiable, and \( g \) and \( h \) can be discontinuous, non-differentiable, and not in closed form. The goal of solving \( P_c \) is to find a constrained local minimum \( x^* \) with respect to \( \mathcal{N}_c(x^*) = \{ x' : \| x' - x^* \| \leq \epsilon \text{ and } \epsilon \to 0 \} \), the continuous neighborhood of \( x^* \), where \( \epsilon \to 0 \) means that \( \epsilon \) is arbitrarily close to 0.

Definition 1. Point \( x^* \) is a C.L.M\(_c\), a constrained local minimum of \( P_c \) with respect to points in \( \mathcal{N}_c(x^*) \), if \( x^* \) is feasible and \( f(x^*) \leq f(x) \) for all feasible \( x \in \mathcal{N}_c(x^*) \).

Traditional Lagrangian theory for continuous optimization works for \( P_c \) with continuous and
differentiable constraint functions $g$ and $h$. The Lagrangian function of $P_c$ with Lagrange-multiplier vectors $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in \mathbb{R}^m$ and $\mu = (\mu_1, \ldots, \mu_r)^T \in \mathbb{R}^r$, is defined as:

$$L(x, \lambda, \mu) = f(x) + \lambda^T h(x) + \mu^T g(x).$$

Under the continuity and differentiability assumptions, a $CLM_c$ satisfies the following necessary KKT condition and sufficient saddle-point condition.

a) Necessary Karush-Kuhn-Tucker (KKT) condition [4]. Assuming $x^*$ is a $CLM_c$ and a regular point, then there exist unique $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^r$ such that:

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0$$

where $\mu_j = 0 \ \forall \ j \notin A(x^*) = \{i \mid g_i(x^*) = 0\}$ (the set of active constraints), and $\mu_j > 0$ otherwise.

The unique $x$, $\lambda$ and $\mu$ that satisfy (4) can be found by solving (4) as a system of nonlinear equations. For instance, for $P_c$ with only equality constraints, the KKT condition in (4) can be expressed as a system of $v + m$ equations in $v + m$ unknowns:

$$F(x, \lambda) = \left[ \nabla_x f(x) + \lambda^T \nabla_x h(x) \right] / h(x) = 0,$$

where $\nabla_x h(x)^T = [\nabla_x h_1(x), \ldots, \nabla_x h_m(x)]$ is the Jacobian of the constraints. The $v + m$ unknowns are solvable when the matrix in (5) is nonsingular.

Iterative procedures have been developed to find the unique $\lambda$, $\mu$ and $x$ that satisfy (4). For example, existing sequential quadratic-programming solvers like SNOPT and LANCELOT solve (4) iteratively by forming a quadratic approximation, evaluating the quadratic model, and updating estimates of $x$, $\lambda$, and $\mu$ until a solution to (4) has been found.

In short, existing CNLP solvers have continuity and differentiability requirements and cannot be applied to solve the type of planning problems studied in this paper.

b) Sufficient saddle-point condition [26, 2]. The concept of saddle points has been studied extensively in the past. For continuous and differentiable constraint functions, $x^*$ is a $CLM_c$ of $P_c$ if there exist unique $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^r$ that satisfy the following saddle-point condition at $x^*$:

$$L(x^*, \lambda, \mu) \leq L(x^*, \lambda^*, \mu^*) \leq L(x, \lambda^*, \mu^*)$$

for all $x \in \mathcal{N}_c(x^*)$ and all $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^r$. This condition is only sufficient but not necessary because there may not exist $\lambda^*$ and $\mu^*$ that satisfy (6) at each $CLM_c$, $x^*$ of $P_c$.

To illustrate the concept, consider the following CNLP with $CLM_c$ at $x^* = 5$:

$$\min_x \quad f(x) = -x^2 \quad \text{subject to} \quad h(x) = x - 5 = 0.$$  

By applying the KKT condition, we differentiate the Lagrangian function $L(x, \lambda) = -x^2 + \lambda(x - 5)$ with respect to $x$ and evaluate it at $x^* = 5$. We have $\nabla_x L(x, \lambda) |_{x^*} = -10 + \lambda = 0$, which implies $\lambda^* = 10$. However, since $\nabla^2_x L(x, \lambda) |_{x^*, \lambda^*} = -2 < 0$, we know that $L(x, \lambda)$ is at a local maximum

---

1Point $x$ is a regular point [29] if gradient vectors of equality constraints $\nabla_x h_1(x), \ldots, \nabla_x h_m(x)$ and active inequality constraints $\nabla_x g_i(x), \ldots, \nabla_x g_i(x), a_i \in A(x)$ (the set of active inequality constraints) are linearly independent. An inequality constraint $g_i(x) \leq 0$ is active when $g_i(x) = 0$. It will affect the search direction only when it is active and can be ignored otherwise.

2Constraint partitioning studied in this paper can be applied to solve problems solvable by existing CNLP and MINLP solvers. This is done by decomposing the constraints of a large CNLP or MINLP problem into subproblems, calling an existing CNLP or MINLP solver to solve the subproblems, and applying constraint-reweighting to resolve violated global constraints. Results on this approach are beyond the scope here and is reported elsewhere [49].

---

*Artificial Intelligence* (accepted to appear)
with respect to $x$ at $(x^*, \lambda^*)$ instead of a local minimum. Hence, there exists no $\lambda^*$ that will allow the second inequality in (6) to be satisfied at $x^* = 5$.

In practice, it is difficult to use (6) for finding the unique $x^*$, $\lambda^*$, and $\mu^*$ that satisfy (4) because it is expressed as a system of nonlinear inequalities that are more difficult to solve than nonlinear equalities. It is mainly used for verifying the solutions found by solving (4).

A recent local optimal method for solving $P_c$ with continuous and differentiable constraint functions is the interior-point $\ell_1$-penalty method based on the following $\ell_1$-penalty function [17]:

$$
\ell_1(z, c) = f(z) + c \cdot \max \left(0, |h_1(z)|, \cdots, |h_m(z)|, g_1(z), \cdots, g_q(z) \right).
$$

Its theory shows that there is a one-to-one correspondence between a CLM$c$ and an unconstrained local minimum of (8) when $c$ is larger than a finite threshold $c^*$. Although it appears that $c$ is not unique, it can be proved that $c^*$ is the maximum of all Lagrange multipliers of the corresponding Lagrangian formulation that satisfies the KKT condition. The approach cannot support constraint partitioning because it is difficult to partition (8) by its constraints and to reach a consistent value of a single penalty term $c$ across the subproblems.

Another partitioning approach called separable partitioning [4] has similar advantages as our proposed constraint partitioning. By exploiting some separable properties in the original problem, these methods decompose the dual problem of $P_c$ with continuous and differentiable constraint functions into multiple much simpler subproblems, each involving only a subset of the constraints and variables. They are limited in their applications because they have restricted assumptions, such as linearity or convexity of the functions.

**Mixed-integer NLP (MINLP) methods** generally solve a MINLP problem by partitioning its search space into subspaces (subproblems) in such a way that, after fixing a subset of the variables, each subproblem is convex and is easily solvable, or can be relaxed and be approximated. There are several approaches.

a) **Generalized Benders decomposition (GBD)** [15] computes in each iteration an upper bound on the solution sought by solving a primal problem and a lower bound on a master problem. Here, the primal problem corresponds to the original problem with fixed discrete variables, and the master problem is derived through nonlinear duality theory. It generally requires the original problem to have special decomposable structures and the subproblems to have some special properties, such as nonempty and continuous subspaces with convex objective and constraint functions.

b) **Outer approximation (OA)** [11] is similar to GBD except that the master problem is formulated using primal information and outer linearization. It requires the continuous subspace to be a nonempty, compact and convex set, and the objective and constraint functions to be convex.

c) **Generalized cross decomposition (GCD)** [19, 20, 39] iterates between a phase solving the primal and dual subproblems and a phase solving the master problem. Similar to OA and GBD, GCD requires the objective and constraint functions of subproblems to be proper convex functions.

d) **Branch and reduce methods** [40, 41] solve MINLPs and CNLPs by a branch-and-bound algorithm and exploit factorable programming to construct relaxed problems as well as range reduction to improve the performance of their bounding procedures. Many of the range-reduction techniques are applicable only when the relaxed problems are convex.

e) **Direct-solution methods** attack a problem without any transformation. They are very limited in handling problems with nonlinear constraints and disconnected feasible regions.

In summary, existing MINLP methods solve a problem either as a whole or by partitioning its variables into subspaces. They are not applicable to solve planning problems due to their convexity or factorability requirement on the decomposed subproblems.

*Artificial Intelligence (accepted to appear)*
Penalty methods. A penalty function of a constrained optimization problem is a summation of its objective function and its constraint functions weighted by penalties. Using penalty vectors \( \alpha \in \mathbb{R}^m \) and \( \beta \in \mathbb{R}^r \), the general penalty function for \( P_c \) is:

\[
L_p(x, \alpha, \beta) = f(x) + \alpha^T P(h(x)) + \beta^T Q(g(x)),
\]

where \( P \) and \( Q \) are possible transformation functions. The goal of a penalty method is to find suitable \( \alpha^* \) and \( \beta^* \) in such a way that \( x^* \) that minimizes (9) corresponds to a \( CLM_c \) of \( P_c \). Penalty methods belong to a general approach that can solve continuous, discrete, and mixed constrained optimization problems, including planning problems, with no continuity, differentiability, and convexity requirements. The Lagrangian function (3) used in the KKT condition is a special case of (9) when \( g(x) \) and \( h(x) \) are continuous differentiable functions that satisfy the regularity condition and are not transformed by \( P \) and \( Q \). When \( P(g(x)) \) and \( Q(h(x)) \) are general functions that take positive and negative values, a local minimum of (9) at \( x^* \) will require finding unique values of \( \alpha^* \) and \( \beta^* \) (proof not shown). However, these unique penalty vectors may either not exist at \( x^* \), or exist but (9) is not at a local minimum at \( x^* \). For instance, for the problem in (7), there is no finite \( \alpha \) that will lead to a local minimum of the penalty function \( L_p(x, \alpha) = f(x) + \alpha \cdot h(x) \) at \( x^* = 5 \). Hence, it will not be possible to find \( x^* \) by minimizing \( L_p(x, \alpha) \) with respect to \( x \) for any given \( \alpha \).

Next, we survey some general results on penalty methods that associate the constrained global minimum of a constrained minimization problem to the global minimum of (9) with sufficiently large penalties. Although we describe the results with respect to continuous problems, they apply to discrete and mixed problems as well.

A static-penalty method \([29, 36]\) formulates \( P_c \) as the minimization of (9) when the constraints of \( P_c \) are transformed into the following non-negative functions: a) \( P(h(x)) \geq 0 \) and \( Q(g(x)) \geq 0 \); and b) \( P(h(x)) = 0 \) if and only if \( h(x) = 0 \), and \( Q(g(x)) = 0 \) if and only if \( g(x) \leq 0 \).

For any finite penalty vectors \( \alpha^{**} \) and \( \beta^{**} \) larger than some thresholds, \( \alpha^{**} \geq \alpha^*3 \) and \( \beta^{**} > \beta^* \), a global minimum \( x^* \) of \( L_p(x, \alpha^{**}, \beta^{**}) \) has a one-to-one correspondence to a constrained global minimum (\( CGM_c \)) of \( P_c \). To show this result, we know that \( \alpha \) and \( \beta \) in (9) must be greater than zero in order to penalize the violated constraints because \( P(h(x)) \) and \( Q(g(x)) \) are non-negative with a minimum of zero. Since (9) is to be minimized with respect to \( x \), increasing the penalty of a violated constraint to a large enough value will force the corresponding transformed constraint function to achieve the minimum of zero, and such penalties always exist if a feasible solution to \( P_c \) exists. At those points where all the constraints are satisfied, every term on the right of (9) except the first is zero, and a global minimum of (9) corresponds to a \( CGM_c \) of \( P_c \).

Continuing on the example in (7), if we use a penalty function that takes the absolute value of the constraint function, namely, \( L_p(x, \alpha) = f(x) + \alpha \cdot |h(x)| \), and assume that \(-100 \leq x \leq 100 \), then there will be a global minimum of \( L_p(x, \alpha^{**}) \) at \( x^* = 5 \) for any \( \alpha^{**} > \alpha^* = 105 \). It is interesting to note that \( \alpha^* \) depends on the range of \( x \). For example, Figure 6 show that, if \(-1000 \leq x \leq 1000 \), then there will be a global minimum of \( L_p(x, \alpha^{**}) \) at \( x^* = 5 \) for any \( \alpha^{**} > \alpha^* = 1005 \). This example shows that \( \alpha^* \) can be exceedingly large in order to ensure global optimality in a given range of \( x \).

One of the difficulties of the static-penalty method is that its penalties have to be found by trial and error. Moreover, each trial is computationally expensive, if not impossible, because it involves finding a global minimum of a nonlinear function. Techniques like simulated annealing \([25]\) can be used, although they only achieve global optimality with asymptotic convergence.

Instead of finding \( \alpha^{**} \) and \( \beta^{**} \) by trial and error, a dynamic-penalty method \([29, 36]\) increases the penalties in (9) gradually, finds the global minimum \( x^{*} \) of (9) with respect to \( x \) for each

\[^{4}\alpha^{**} > \alpha^* \text{ means that every element of } \alpha^{**} \text{ is larger than the corresponding element of } \alpha^*.\]
2.2 Existing Mathematical Programming Methods

unconstrained problem in the sequence, and stops when \( x^* \) is a feasible solution to \( P_c \). To show that \( x^* \) is a \( CGM_c \) when the algorithm stops, we know that the penalties need to be increased when \( x^* \) is a global minimum of (9) but not a feasible solution to \( P_c \). The first time \( x^* \) is a feasible solution to \( P_c \), the solution must also be a \( CGM_c \). Hence, the method leads to the smallest \( \alpha^{**} \) and \( \beta^{**} \) that allows a \( CGM_c \) to be found. However, it has the same limitation as the static-penalty method because it requires finding global minima of nonlinear functions.

The practice of re-weighting violated constraints during a local search of penalty functions has been popular and highly successful in the AI community. For example, planners such as SATPLAN [22], Blackbox [23], and ILP-PLAN [24] first transform a planning problem into a SAT or an ILP (integer linear programming) formulation. They then find a solution to the SAT or ILP problem using an existing solver that minimizes a penalty function of the form in (9) with dynamically adjusted penalties [43, 44, 42, 24]. The key feature in these applications is that they deal with discrete constraint functions that are non-negative to start with, such as the number of violated clauses in a problem and binary constraints on whether a clause is violated. Hence, they work well without the need to transform the constraint functions. Moreover, the objective function is usually chosen in such a way that finding a constrained local minimum amounts to finding a constrained global minimum of the constrained SAT model. As a result, the theory of existing static and dynamic penalty methods applies.

In short, using (9) when \( P(h(x)) \) and \( Q(g(x)) \) can take positive and negative values, a \( CLM_c \) \( x^* \) of \( P_c \) does not imply a local minimum of (9) at \( x^* \) because there may not exist feasible penalties there. This means that a \( CLM_c \) whose penalties do not exist in (9) cannot be found by looking for a local minimum of (9). On the other hand, using (9) when \( P(h(x)) \) and \( Q(g(x)) \) are non-negative functions, a \( CGM_c \) of \( P_c \) always corresponds to an unconstrained global minimum of (9) when its penalties are larger than some thresholds. Unfortunately, this result is impractical because finding a global minimum of an unconstrained nonlinear function is computationally expensive. A similar observation can be made on discrete and mixed optimization problems.

To cope with these shortcomings, we prove in the next section the one-to-one correspondence between a constrained local minimum of a MINLP and an extended saddle point of its penalty function with non-negative (transformed) constraint functions, when the penalties larger than some thresholds. This result extends our previous work that proves a special-case condition for discrete optimization problems [51]. A constrained local minimum of a MINLP can, therefore, be found by increasing gradually the penalties of those violated constraints and by looking for a local minimum of the penalty function using any existing algorithm until a solution to the constrained model is found. By showing a general theory that covers continuous, discrete and mixed-integer optimization, this paper provides a complete foundation on penalty methods.

Figure 6: An illustration of a global minimum of \( L_p(x, \alpha^{**}) \) at \( x^* = 5 \) for any \( \alpha^{**} > \alpha^* = 1005 \) but not one when \( \alpha^{**} < \alpha^* \) for the CNLP problem in (7), where \(-1000 \leq x \leq 1000\).
3 The Theory of Extended Saddle Points

We describe in this section our necessary and sufficient saddle-point condition (ESPC) in mixed space based on a penalty function with non-negative (transformed) constraint functions and under a relaxed range of penalties.

3.1 ESPC for Continuous, Discrete, and Mixed Optimization

We first state the necessary and sufficient ESPC on $CLM_c$ of $P_c$, based on the following penalty function.

**Definition 2.** The penalty function for $P_c$ in (2) is defined as in (9) by transforming the constraint functions of $P_c$ into non-negative functions:

$$L_c(x, \alpha, \beta) = f(x) + \alpha^T |h(x)| + \beta^T \max(0, g(x)),$$

where $|h(x)| = (|h_1(x)|, \ldots, |h_m(x)|)^T$ and $\max(0, g(x)) = (\max(0, g_1(x)), \ldots, \max(0, g_r(x)))^T$; and $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^r$ are penalty vectors. Note that (10) is a special case of (9).

In continuous space, we need the following constraint-qualification condition in order to establish the existence of a local minimum of (10) at $x^*$.

**Definition 3.** The subdifferential $D_x(\phi(x'), \bar{p})$ of function $\phi$ at $x' \in X$ along direction $\bar{p} \in X$ represents the rate of change of $\phi(x')$ under an infinitely small perturbation along $\bar{p}$. That is,

$$D_x(\phi(x'), \bar{p}) = \lim_{\epsilon \to 0} \frac{\phi(x' + \epsilon \bar{p}) - \phi(x')}{\epsilon}. \quad (11)$$

Note that a function whose subdifferential exists along $\bar{p}$ at $x'$ does not imply that $\phi(x)$ is differentiable at $x'$ with respect to $\bar{p}$.

**Definition 4.** Constraint-qualification condition. The solution $x^* \in X$ of $P_c$ meets the condition if there exists no direction $\bar{p} \in X$ along which the subdifferentials of continuous equality and active continuous inequality constraints are all zero. That is,

$$\exists \bar{p} \in X \text{ such that } D_x(h_i(x^*), \bar{p}) = 0 \text{ and } D_x(g_j(x^*), \bar{p}) = 0 \quad \forall i \in C_h \text{ and } j \in C_g, \quad (12)$$

where $C_h$ and $C_g$ are, respectively, the sets of indexes of continuous equality and active continuous inequality constraints. Constraint qualification is always satisfied if $C_h$ and $C_g$ are empty sets.

Our constraint-qualification condition requires the subdifferential of at least one continuous equality constraint or active continuous inequality constraint at $x^*$ to be non-zero along each and every direction $\bar{p}$. It rules out the case in which there is a direction $\bar{p}$ at $x^*$ along which all equality constraints and active inequality constraints have zero subdifferentials. Intuitively, constraint qualification at $x^*$ ensures the existence of finite $\alpha$ and $\beta$ that lead to a local minimum of (10) at $x^*$. Consider a neighboring point $x^* + \bar{p}$ infinitely close to $x^*$, where the objective function $f$ at $x^*$ decreases along $\bar{p}$ and all active constraints at $x^*$ have zero subdifferentials along $\bar{p}$. In this case, since all the active constraints at $x^* + \bar{p}$ are also satisfied, it will be impossible to find finite $\alpha$ and $\beta$ in order to establish a local minimum of (10) at $x^*$ with respect to $x^* + \bar{p}$. To ensure a local minimum of (10) at $x^*$, the above scenario must not be true for any $\bar{p}$. Note that our condition is less restricted than the regularity condition in KKT, which requires the linear independence of gradients of equality and active inequality constraint functions.

The following theorem states the ESPC when the constraint qualification is satisfied.

_Artificial Intelligence_ (accepted to appear)
3.1 ESPC for Continuous, Discrete, and Mixed Optimization

**Theorem 1.** Necessary and sufficient ESPC on CLM$_c$ of $P_c$. Suppose $x^* \in \mathbb{R}^n$ is a point in the continuous search space of $P_c$ and satisfies the constraint-qualification condition in (12), then $x^*$ is a CLM$_c$ of $P_c$ if and only if there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that the following is satisfied:

$$L_c(x^*, \alpha, \beta) \leq L_c(x^*, \alpha^{**}, \beta^{**}) \leq L_c(x, \alpha^{**}, \beta^{**})$$

for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$, and for all $x \in \mathcal{N}_c(x^*)$, $\alpha \in \mathbb{R}^m$, and $\beta \in \mathbb{R}^r$.

The proof of the theorem is shown in Appendix A.

Theorem 1 shows that $x^*$, a local minimum of (10) with respect to $x$, corresponds to a CLM$_c$ of $P_c$ (second inequality of (13)) when $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$ such that all the constraints of $P_c$ are forced to be satisfied (first inequality of (13)). Hence, instead of looking for CLM$_c$'s directly, it suffices to look for extended saddle points in the penalty formulation.

According to (13), an extended saddle point is a local minimum of $L_c$ with respect to $x$ and a local maximum of $L_c$ with respect to $\alpha$ and $\beta$. One approach to look for an extended saddle point of $L_c$ is to increase gradually $\alpha^{**}$ and $\beta^{**}$, while minimizing $L_c$ with respect to $x$ using an existing local-search method, until $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. Because there are many existing local-search algorithms, our approach improves over the static-penalty approach, which is defined with respect to difficult-to-find global minima of (9). However, as presented in Theorem 4 later, our approach only generates fixed points that are necessary, but not sufficient, to be CLM$_c$. Additional steps presented in Section 3.2 are needed to allow our approach to find CLM$_c$.

It is interesting to note that $\alpha^*$ and $\beta^*$ that satisfy Theorem 1 can be much smaller than the corresponding $\alpha^*$ and $\beta^*$ found by the dynamic-penalty method in Section 2.2. Continuing on the example in (7), instead of $\alpha^{**} > \alpha^* = 1005$ in order to have a global minimum of $L_p(x, \alpha^{**})$ at $x^* = 5$ for $-1000 \leq x \leq 1000$ in the dynamic-penalty method (Figure 6), it suffices to have $\alpha^{**} > \alpha^* = 10$ by applying Theorem 1 in order to have a local minimum of $L_c(x, \alpha^{**}) = -x^2 + \alpha^{**} |x-5|$ at $x^* = 5$, irrespective of the range of $x$. Figure 7 illustrates that $L_c(x, \alpha^{**})$ is at a local minimum around $x^* = 5$ when $\alpha^{**} = 20$ but is not one when $\alpha^{**} = 10$. A small $\alpha^{**}$ leads to a less rugged $L_c(x, \alpha^{**})$ function and makes it easier for global-search algorithms to locate local minima.

Next, we present the ESPC of discrete nonlinear programming (DNLP) problems. Consider the DNLP whose $f$, $g$ and $h$ are not necessarily continuous and differentiable with respect to $y$.

$$(P_d): \quad \min_y f(y) \quad \text{where } y = (y_1, \ldots, y_w)^T \in \mathbb{D}^w$$

subject to $h(y) = 0$ and $g(y) \leq 0$.  

The goal of solving $P_d$ is to find a constrained local minimum $y^*$ with respect to $\mathcal{N}_d(y^*)$, the discrete neighborhood of $y^*$. Since the discrete neighborhood of a point is not well defined in the

*Artificial Intelligence* (accepted to appear)
literature, it is up to the user to define the concept. Intuitively, $\mathcal{N}_d(y)$ represents points that are perturbed from $y$, with no requirement that there be valid state transitions from $y$.

**Definition 5.** $\mathcal{N}_d(y)$ [1], the discrete neighborhood of $y \in \mathbb{D}^w$ in discrete space, is a finite user-defined set of points \{y' \in \mathbb{D}^w\} such that y' is reachable from y in one step, that $y' \in \mathcal{N}_d(y) \iff y \in \mathcal{N}_d(y')$, and that it is possible to reach every $y''$ from any $y$ in one or more steps through neighboring points.

**Definition 6.** Point $y^*$ is a CLM$_d$, a constrained local minimum of $P_d$ with respect to points in $\mathcal{N}_d(y^*)$, if $y^*$ is feasible and $f(y^*) \leq f(y)$ for all feasible $y \in \mathcal{N}_d(y^*)$.

There are two distinct features of CLM$_d$. First, the set of CLM$_d$ of $P_d$ is neighborhood dependent, and a point may be a CLM$_d$ under one definition of neighborhood but may not be one under another. However, all CLM$_d$’s are guaranteed to be feasible, even in the extreme case in which the neighborhood of each point includes only itself. The fact that CLM$_d$’s are neighborhood dependent is not critical in constrained searches, because our goal is to find feasible solutions that are better than their neighboring points. As long as a consistent neighborhood is used throughout a search, a CLM$_d$ found will be a local minimum with respect to its neighborhood. Second, a discrete neighborhood has a finite number of points. Hence, the verification of a point to be a CLM$_d$ can be done by comparing its objective value against that of its finite number of neighbors. This feature allows the search of a descent direction in discrete space to be done by enumeration or by greedy search.

**Definition 7.** The penalty function for $P_d$ is defined as in (9) by transforming the constraint functions of $P_d$ into non-negative functions:

$$L_d(y, \alpha, \beta) = f(y) + \alpha^T h(y) + \beta^T \max(0, g(y)) \quad \text{where } \alpha \in \mathbb{R}^m \text{ and } \beta \in \mathbb{R}^r. \quad (15)$$

**Theorem 2.** Necessary and sufficient ESPC on CLM$_d$ of $P_d$ [55, 51]. Suppose $y^* \in \mathbb{D}^w$ is a point in the discrete search space of $P_d$. Then $y^*$ is a CLM$_d$ of $P_d$ if and only if there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that the following condition is satisfied:

$$L_d(y^*, \alpha^*, \beta^*) \leq L_d(y^*, \alpha^{**}, \beta^{**}) \leq L_d(y, \alpha^{**}, \beta^{**}) \quad (16)$$

for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$, and for all $y \in \mathcal{N}_d(y^*)$, $\alpha \in \mathbb{R}^m$, and $\beta \in \mathbb{R}^r$.

The proof of the theorem is shown in Appendix B.

Note that the constraint-qualification condition in Theorem 1 is not needed in Theorem 2 because constraint functions are not changing continuously in discrete problems.

Last, we present the ESPC of MINLP problems. Consider a MINLP problem whose objective function $f$ is continuous and differentiable with respect to the continuous subspace $x$:

$$(P_m): \min_{x, y} f(x, y) \quad \text{where } x = (x_1, \ldots, x_v)^T \in \mathbb{R}^v \text{ and } y = (y_1, \ldots, x_w)^T \in \mathbb{D}^w \quad (17)$$

subject to $h(x, y) = 0$ and $g(x, y) \leq 0$.

The goal of solving $P_m$ is to find a constrained local minimum $(x^*, y^*)$ with respect to $N_m(x^*, y^*)$, the mixed neighborhood of $(x^*, y^*)$. In this paper, we construct our mixed neighborhood as the union of points perturbed in either the discrete or the continuous subspace, but not both. Such a definition allows the ESPC for the two subspaces to be decomposable into that for each subspace. Note that a mixed neighborhood is also a user-defined concept because a discrete neighborhood is user-defined and a mixed neighborhood is a union of discrete and continuous neighborhoods.

*Artificial Intelligence* (accepted to appear)
3.2 Search Procedures for Finding Extended Saddle Points

As is discussed in the last section, a CLM* of \( P_c \) can be found by gradually increasing \( \alpha^{**} \) and \( \beta^{**} \), while minimizing \( L_c(x, \alpha^{**}, \beta^{**}) \), until \( \alpha^{**} > \alpha^* \) and \( \beta^{**} > \beta^* \). This observation allows us to solve \( P_c \) by an iterative search in Figure 8a. (The algorithm for solving \( P_d \) is similar and is not shown.)

Assuming \( \alpha^{**} \) and \( \beta^{**} \) have been found in the outer loop and according to the second inequality in (13), the inner loop looks for a local minimum of \( L_c(x, \alpha^{**}, \beta^{**}) \) in order to find \( x^* \). If a feasible
3.2 Search Procedures for Finding Extended Saddle Points

procedure ESP\textsubscript{search\_continuous}(P\textsubscript{c}, x, α\textsuperscript{max}, β\textsuperscript{max});
\begin{verbatim}
α ← 0; β ← 0;
repeat
  for (i = 1, \ldots, m) if (h\textsubscript{i}(x) \neq 0 and α\textsubscript{i} < α\textsuperscript{i}\textsubscript{max}) then increase α\textsubscript{i} by δ;
  for (j = 1, \ldots, r) if (g\textsubscript{j}(x) \neq 0 and β\textsubscript{j} < β\textsuperscript{j}\textsubscript{max}) then increase β\textsubscript{j} by δ;
  repeat
    perform descent of L\textsubscript{c}(x, α, β) with respect to x;
  until a local minimum of L\textsubscript{c}(x, α, β) is found;
  until (α\textsubscript{i} > α\textsuperscript{i}\textsubscript{max} for all h\textsubscript{i}(x) \neq 0 and β\textsubscript{j} > β\textsuperscript{j}\textsubscript{max} for all g\textsubscript{j}(x) < 0)
  return CLM\textsubscript{c} of P\textsubscript{c} if found;
end procedure
\end{verbatim}

a) Direct implementation of (13) to look for CLM\textsubscript{c} of P\textsubscript{c} for given starting point x

procedure ESP\textsubscript{search\_mixed}(P\textsubscript{m}, z, α\textsuperscript{max}, β\textsuperscript{max});
\begin{verbatim}
α ← 0; β ← 0;
repeat
  for (i = 1, \ldots, m) if (h\textsubscript{i}(z) \neq 0 and α\textsubscript{i} < α\textsuperscript{i}\textsubscript{max}) then increase α\textsubscript{i} by δ;
  for (j = 1, \ldots, r) if (g\textsubscript{j}(z) \neq 0 and β\textsubscript{j} < β\textsuperscript{j}\textsubscript{max}) then increase β\textsubscript{j} by δ;
  repeat
    perform descent of L\textsubscript{m}(z, α, β) with respect to x for given y;
  until a local minimum of L\textsubscript{m}(z, α, β) is found;
  repeat
    perform descent of L\textsubscript{m}(z, α, β) with respect to y for given x;
  until a local minimum of L\textsubscript{m}(z, α, β) is found;
  until (α\textsubscript{i} > α\textsuperscript{i}\textsubscript{max} for all h\textsubscript{i}(z) \neq 0 and β\textsubscript{j} > β\textsuperscript{j}\textsubscript{max} for all g\textsubscript{j}(z) < 0)
  or a CLM\textsubscript{m} of P\textsubscript{m} is found;
return CLM\textsubscript{m} if found;
end procedure
\end{verbatim}

b) Direct implementation of (21) and (22) to look for CLM\textsubscript{m} of P\textsubscript{m} for given starting point z = (x, y)

Figure 8: Iterative procedures to look for CLM\textsubscript{c} of P\textsubscript{c} and CLM\textsubscript{m} of P\textsubscript{m}. The bounds on α and β, α\textsuperscript{max} and β\textsuperscript{max}, are user-provided.

solution to P\textsubscript{c} is not found at the local minimum x of L\textsubscript{c}(x, α\textsuperscript{**}, β\textsuperscript{**}), the penalties corresponding to the violated constraints are increased. The process is repeated until a CLM\textsubscript{c} is found or when α\textsuperscript{**} (resp. β\textsuperscript{**}) is larger than the user-provided maximum bound α\textsuperscript{max} (resp. β\textsuperscript{max}), where α\textsuperscript{max} (resp. β\textsuperscript{max}) is chosen to be so large that it exceeds α\textsuperscript{*} (resp. β\textsuperscript{*}).

Figure 8b shows the pseudo code which solves P\textsubscript{m} by looking for x\textsuperscript{*}, y\textsuperscript{*}, α\textsuperscript{**}, and β\textsuperscript{**} that satisfy Corollary 1. By performing descents of L\textsubscript{m}(x, y, α, β) in the continuous and discrete neighborhoods in the two inner loops, it looks for a local minimum (x\textsuperscript{*}, y\textsuperscript{*}) of L\textsubscript{m}(x, y, α, β) with respect to points in N\textsubscript{m}(x, y). The outer loop increases the penalties of violated constraints and stops when a CLM\textsubscript{m} is found or when α\textsuperscript{**} (resp. β\textsuperscript{**}) exceeds its maximum bound α\textsuperscript{max} (resp. β\textsuperscript{max}).

Because L\textsubscript{c}(x, α\textsuperscript{**}, β\textsuperscript{**}) and L\textsubscript{m}(x, y, α\textsuperscript{**}, β\textsuperscript{**}) may have many local minima and some of them do not correspond to constrained local minima even when α\textsuperscript{**} > α\textsuperscript{*} and β\textsuperscript{**} > β\textsuperscript{*}, it is possible for the iterative procedures in Figure 8 to terminate without finding a constrained local minimum. The following theorem summarizes this observation.

**Theorem 4.** When α\textsuperscript{max} > α\textsuperscript{*} and β\textsuperscript{max} > β\textsuperscript{*}, the iterative procedure in Figure 8a (resp. 8b) generates fixed points that are necessary but not sufficient to satisfy (13) (resp. (21) and (22)).

To cope with this issue, we discuss three additional strategies to augment the procedure in
Figure 8b. These strategies are general and are applicable when looking for $CLM_c$ and $CLM_d$.

First, when $\alpha^{**}$ and $\beta^{**}$ reach their upper bounds during a search but a local minimum of $L_m(x, y, \alpha^{**}, \beta^{**})$ does not correspond to a $CLM_m$ of $P_m$, then a different local minimum of the function will need to be found. Instead of restarting the search from a new starting point, reducing $\alpha^{**}$ and $\beta^{**}$ will change the terrain and “lower” the barrier of the penalty function, thereby allowing a local search to continue on the same trajectory and move to another local minimum of the penalty function. By repeatedly increasing $\alpha^{**}$ and $\beta^{**}$ to their upper bounds and reducing them to some lower bounds, a local search algorithm will be able to visit multiple local minima of the penalty function. Alternatively, it is possible to escape from a local minimum of the penalty function by using a global-search algorithm in the inner loops. Since these two strategies offset each other in their effects, only one of them will need to be applied.

Second, the ease of finding a $CLM_m$ depends on the number of $CLM_m$’s in the search space of $P_m$, which in turn depends on the neighborhood function chosen. If the neighborhood of each point is the entire search space, then finding a $CLM_m$ amounts to finding a constrained global minimum. On the other hand, if the neighborhood of each point is only the point itself, then any feasible point in the search space is a $CLM_m$. In this case, since the neighborhood is limited, only random probing can be applied, and finding a $CLM_m$ amounts to feasibility search. In practice, we choose the neighborhood of each point to be rich enough in order to achieve a balance between the number of neighbors of each point and the number of $CLM_m$’s in the search space.

Last, because functions in planning problems may not be in closed form and their gradients are unavailable, it is hard to locate local minima of the penalty function in this case. One way to address this issue is to generate probes based on deterministic, probabilistic, or genetic mechanisms and accept them based on some deterministic or stochastic criteria. For example, in our experiments in Section 5.1 on using ASPEN to solve subproblems, new probes generated using ASPEN’s built-in mechanism during the descent of the penalty function are accepted based on the Metropolis probability when $L_d$ increases. This mechanism allows descents as well as occasional ascents of the penalty function. In more general cases, as is illustrated in the stochastic constrained simulated annealing algorithm [50], new probes generated are accepted based on the Metropolis probability when $L_m$ increases along one of the $x$ or $y$ dimension and decreases along the $\alpha$ or $\beta$ dimension.

4 Partitioning of ESPC for Temporal Planning Problems

In this section, we solve $P_t$ in (1) by finding plan $z$ that is a $CLM_m$ with respect to feasible plans in its mixed neighborhood $\mathcal{N}_m(z)$. After showing that $z$ satisfies the ESPC in (20), we decompose the ESPC into a set of necessary conditions that collectively are sufficient. Problem $P_t$ is then solved by iteratively finding an extended saddle point in each stage and by resolving those violated global constraints using appropriate penalties.

4.1 Necessary and Sufficient ESPC for Partitioned Subproblems

To simplify our discussion, we do not partition plan $z$ into discrete and continuous parts in the following derivation, although it is understood that each partition will need to be further decomposed in the same way as in Corollary 1. To enable the partitioning of the ESPC into independent necessary conditions, we define a mixed neighborhood of plan $z$ as follows:

Artificial Intelligence (accepted to appear)
Definition 11. \( N_p(z) \), the mixed neighborhood of \( z \) for partitioned problem \( P_t \), is defined as:
\[
N_p(z) = \bigcup_{t=0}^{N} N^{(t)}_p(z) = \bigcup_{t=0}^{N} \left\{ z' \mid z'(t) \in N_m(z(t)), \text{ and } z'_i(s) = z_i(s) \forall z_i(s) \notin z(t), s \neq t, i = 1, \ldots, u_s \right\},
\]
where \( N_m(z(t)) \) is the mixed-space neighborhood of \( z(t) \) in Stage \( t \).

Intuitively, \( N_p(z) \) is decomposed into \( N + 1 \) neighborhoods, each perturbing \( z \) in only one of the stages of \( P_t \), while keeping the overlapped variables consistent in the other stages. The size of \( N_p(z) \) defined in (23) is smaller than the Cartesian product of the neighborhoods across all stages.

By considering \( P_t \) as a MINLP and by defining the corresponding penalty function, we can apply Theorem 3 as follows.

Definition 12. Let \( \Phi(z, \gamma, \eta) = \gamma^T|H(z)| + \eta^T \max(0, G(z)) \) be the sum of the transformed global constraint functions weighted by their penalties, where \( \gamma = (\gamma_1, \ldots, \gamma_p)^T \in \mathbb{R}^p \) and \( \eta = (\eta_1, \ldots, \eta_q)^T \in \mathbb{R}^q \) are the penalty vectors for the global constraints. Then the penalty function for \( P_t \) and the corresponding penalty function in Stage \( t \) are defined as in (9) by transforming the constraint functions of \( P_t \) into non-negative functions:
\[
L_m(z, \alpha, \beta, \gamma, \eta) = J(z) + \sum_{t=0}^{N} \left\{ \alpha(t)^T h^{(t)}(z(t)) + \beta(t)^T \max(0, g^{(t)}(z(t))) \right\} + \Phi(z, \gamma, \eta),
\]
(23)
\[
\Gamma_m(z, \alpha(t), \beta(t), \gamma, \eta) = J(z) + \alpha(t)^T h^{(t)}(z(t)) + \beta(t)^T \max(0, g^{(t)}(z(t))) + \Phi(z, \gamma, \eta),
\]
(24)
where \( \alpha(t) = (\alpha_1(t), \ldots, \alpha_{m_t}(t))^T \in \mathbb{R}^{m_t} \) and \( \beta(t) = (\beta_1(t), \ldots, \beta_{r_t}(t))^T \in \mathbb{R}^{r_t} \) are the penalty vectors for the local constraints in Stage \( t \).

Lemma 1. Plan \( z \) is a \( CLM_m \) of \( P_t \) with respect to \( N_p(z) \) if and only if there exist finite \( \alpha^* \geq 0, \beta^* \geq 0, \gamma^* \geq 0 \), and \( \eta^* \geq 0 \) such that the following ESPC is satisfied:
\[
L_m(z^*, \alpha, \beta, \gamma, \eta) \leq L_m(z^{*}, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \leq L_m(z, \alpha^{*}, \beta^{**}, \gamma^{**}, \eta^{**})
\]
(25)
for any \( \alpha^{**} > \alpha^*, \beta^{**} > \beta^*, \gamma^{**} > \gamma^* \) and \( \eta^{**} > \eta^* \), and for all \( \alpha \in \mathbb{R}^{\sum_{i=0}^{m_t} m_i}, \beta \in \mathbb{R}^{\sum_{i=0}^{r_t} r_i}, \gamma \in \mathbb{R}^p, \eta \in \mathbb{R}^q, \) and \( z \in N_p(z^*) \).

Based on Lemma 1, we next show the partitioning of (25) into multiple conditions.

Theorem 5. Partitioned necessary and sufficient ESPC on \( CLM_m \) of \( P_t \). Given \( N_p(z) \), the ESPC in (25) can be rewritten into \( N + 2 \) necessary conditions that, collectively, are sufficient:
\[
\Gamma_m(z^*, \alpha(t), \beta(t), \gamma^{**}, \eta^{**}) \leq \Gamma_m(z^{*}, \alpha(t)^{*}, \beta(t)^{*}, \gamma^{**}, \eta^{**}) \leq \Gamma_m(z, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**})
\]
(26)
\[
L_m(z^*, \alpha^*, \beta^{**}, \gamma, \eta) \leq L_m(z^{*}, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**})
\]
(27)
for all \( z \in N_p(z^*) \), \( \alpha(t) \in \mathbb{R}^{m_t}, \beta(t) \in \mathbb{R}^{r_t} \), and \( t = 0, \ldots, N \).

The proof is shown in Appendix D.

Theorem 5 shows that the ESPC in (25) can be partitioned into \( N + 1 \) necessary conditions in (26) on the local constraints and an overall necessary condition in (27) on the global constraints.
4.2 The Partition-and-Resolve Procedure

Figure 9: The partition-and-resolve procedure to look for $CLM_m$ of $P_t$. The bounds $\alpha_{\max}$, $\beta_{\max}$, $\gamma_{\max}$, and $\eta_{\max}^t$ are user-provided.

across the subproblems. A close examination shows that the local extended saddle points in Stage $t$ that satisfy (26) are the local minima of (24) with respect to $z$ (the second inequality of (26)), when $\alpha(t)^{**}$ and $\beta(t)^{**}$ are larger than some thresholds $\alpha(t)^*$ and $\beta(t)^*$ such that all the constraints in Stage $t$ are forced to be satisfied (the first inequality of (26)). In essence, a point that satisfies (26) in Stage $t$ is a solution to the following MINLP $P(t)^{(t)}$, where the original objective function $J(z)$ is biased by the violated global constraints:

$$
\begin{align*}
\min_{z(t)} & \quad J(t)(z) = J(z) + \gamma^T H(z) + \eta^T \max(0, G(z)) \\
\text{subject to} & \quad h(t)(z(t)) = 0 \quad \text{and} \quad g(t)(z(t)) \leq 0.
\end{align*}
$$

The bias on the violated global constraints when solving $P(t)^{(t)}$ is important because it leads the search towards points that minimize this bias. When the penalties on the violated global constraints are large enough, solving $P(t)^{(t)}$ will lead to points, if they exist, that satisfy the global constraints.

In short, finding points that satisfy (25) can be reduced to solving multiple MINLPs defined by $P(t)^{(t)}$ in (28), and to the reweighting of violated global constraints defined in (27).

4.2 The Partition-and-Resolve Procedure

Figure 9 presents the partition-and-resolve procedure, which looks for points that satisfy the conditions in Theorem 5. The inner loop of Stage $t$ in Figure 9b solves $P(t)^{(t)}$ by looking for an extended saddle point that satisfies (26). This can be done by the procedure in Figure 8b, using fixed $\gamma$ and $\eta$ specified in the outer loop, or by an existing solver. The latter is possible because $P(t)^{(t)}$ is a well-defined MINLP. This is illustrated in Section 5 where we use the ASPEN and the MIPS planners to solve the partitioned planning subproblems. After solving the subproblems, the penalties on those violated global constraints are increased in the outer loop. The process is repeated until a $CLM_m$ to $P_t$ has been found or when $\gamma$ and $\eta$ exceed their maximum bounds. Similar to the
result in Theorem 4, the procedure in Figure 9 generates fixed points that are necessary but not sufficient to satisfy (26) and (27). Hence, additional steps described in Section 3.2 are needed to help escape from local minima of the penalty function that are not feasible points to \( P_t \).

5 Experimental Results

In this section, we describe our experimental results on using the discrete-space ASPEN [9] and the mixed-space MIPS [12] planners to solve partitioned planning benchmarks. We show significant improvements in their solutions, both in terms of the quality of the plans generated and the execution times to find them.

5.1 SGPlan\(_t\)(ASPEN): A Planner Using ASPEN to solve Partitioned Problems

We describe the ASPEN (Automated Scheduling and Planning Environment) system [9] developed at the Jet Propulsion Laboratory and its available benchmarks on spacecraft operation planning. We then present our prototype planner SGPlan\(_t\)(ASPEN,N,repartitioning\_strategy) that partitions a problem along its temporal horizon into \( N \) subproblems of the form in \( P_t^{(t)} \), that calls ASPEN to solve the subproblems, that resolves the violated global constraints, and that repartitions the problem if necessary. Finally, we compare the performance between ASPEN and SGPlan\(_t\)(ASPEN,N,repartitioning\_strategy).

ASPEN [9] is an objective-based planning system for the automated planning and scheduling of complex spacecraft operations. It involves generating a sequence of parallel low-level spacecraft commands from a set of high-level science and engineering goals.

Using a discrete time horizon and a discrete state space, an ASPEN model encodes spacecraft operability constraints, flight rules, spacecraft hardware models, science experiment goals, and operations procedures. It defines various types of schedule constraints that may be in procedural form among or within the parallel activities to be scheduled. Such constraints include temporal, decomposition, resource, state-dependency, and goal constraints. In addition, the quality of a schedule is defined by a preference score, which is a weighted sum of multiple preferences (that may also be procedural) to be optimized by the planner. Preferences can be related to the number of conflicts, the number of actions, the value of a resource state, or the value of an activity parameter.

Since ASPEN cannot search for feasible plans and optimize plan quality at the same time, it alternates between a repair phase and an optimization phase. In the repair phase [35], ASPEN generates an initial schedule that may have conflicts and searches for a feasible plan from this initial plan, using iterative repairs to resolve conflicts individually. In a repair iteration, the planner must decide at each choice point a conflict to be resolved and a conflict-resolution method from a rich collection of repair heuristics. In the optimization phase, ASPEN uses a preference-driven, incremental, local-optimization method to optimize plan quality defined by the preference score. It decides the best search direction at each choice point, based on information from multiple choice points. In our experiments, we allow ASPEN to alternate between a repair phase with an unlimited number of iterations and an optimization phase with 200 iterations (both defaults in ASPEN).

The ASPEN software can be tested on several publicly available benchmarks on scheduling parallel spacecraft operations. In this paper, we have tested all the four available benchmarks in the public domain. a) The CX1-PREF benchmark [53] models the planning of operations of the Citizen Explorer-1 (CX-1) satellite that involve taking data related to ozone and downloading the data to ground for scientific analysis. Its problem generator can generate problem instances with a user-specified number of satellite orbits. In our experiments, we have studied CX1-PREF with 8 and 16 orbits, respectively. b) The DCAPS benchmark [34] models the operation of DATA-CHASER...
1. **procedure** SGPlan\_t(ASPEN,N,repartitioning\_strategy)
2. generate initial plan and set initial temperature $T$;
3. partition time horizon into $N$ stages;
4. **repeat**
5. $\text{num\_descents} \leftarrow 1$;
6. **for** $t = 1$ to $N$
7. $\text{for } k = 1$ to $\text{num\_descents}$
8. call ASPEN to solve $P(t)$ in a child process and to generate a new schedule;
9. evaluate $\Gamma_m(z, \alpha(t), \beta(t), \gamma, \eta)$ and the Metropolis probability controlled by $T$;
10. if $\Gamma_m(z, \alpha(t), \beta(t), \gamma, \eta)$ is accepted **then**
11. call ASPEN to apply the action in the main process;
12. update penalties $\alpha(t)$ and $\beta(t)$ on violated local constraints;
13. **end if**
14. **end for**
15. **end for**
16. update penalties $\gamma$ and $\eta$ on violated global constraints;
17. $\text{num\_descents} \leftarrow \min(100, \text{num\_descents} * 2)$;
18. reduce temperature $T \leftarrow T \cdot c$ where $c \in (0,1)$;
19. if (repartitioning\_strategy is DYN\_P) **then** repartition the stages **end if**;
20. **until** no change in $z, \alpha, \beta, \gamma, \eta$ in an iteration;
21. **return** the best plan found;
22. **end procedure**

Figure 10: SGPlan\_t(ASPEN,N,repartitioning\_strategy): The partition-and-resolve procedure used in SG-Plan that partitions a planning problem along its temporal horizon into $N$ subproblems, that calls ASPEN to solve the subproblems, that resolves the violated global constraints, and that repartitions the problem if necessary. Annealing (Lines 9-10) is used to probabilistically accept a probe with worse penalty-function value during descents of $\Gamma_m$.

Implementation of the partition-and-resolve search. Based on Figure 9, we have implemented SGPlan\_t(ASPEN,N,repartitioning\_strategy) [8]. In our implementation, we set the weight of $J(z)$ in $P(t)$ to 100 (since the preference score is between 0 to 1), initialize all penalties to zeros, and increase the penalties of violated global constraints in each iteration by 0.1.

In generating a new schedule from the current schedule during descents of $\Gamma_m$ (Line 8 of Figure 10), ASPEN chooses probabilistically among its repair and optimization actions, selects a random feasible action at each choice point, and applies the selected actions to the current schedule. Since many of the objectives and constraints in complex spacecraft applications are not differentiable, the new schedule generated does not likely follow descent directions, and a local search may get stuck easily in local minima of the penalty function that are not feasible solutions to the original problem. To this end, SGPlan\_t(ASPEN,N,repartitioning\_strategy) employs annealing to determine whether to accept the new schedule (Lines 9-10). Using a parameter called *temperature*, it accepts the new schedule with larger $\Gamma_m$ based on the Metropolis probability, with the acceptance probability decreasing as the temperature decreases ($c \in (0,1)$). In our implementation, we fix the initial temperature to 1,000 and reduce it in every iteration by a factor $c = 0.8$.

Two other important issues that must be addressed in our partition-and-resolve implementation are the number of stages used and the duration of each. In ASPEN, a conflict has an active window bounded by a start time and an end time called the *time points*. Adjacent time points can be collapsed into a stage, since ASPEN has discrete time horizons.
We have studied both the static and the dynamic partitioning of stages. In static partitioning, SGPlan\textsubscript{t}(ASPEN,N,STATIC\textsubscript{P}) partitions the horizon statically and evenly into \(N\) stages. This simple strategy often leads to an unbalanced number of time points in different stages. During a search, some stages may contain no conflicts to be resolved, while others may contain a lot of conflicts. Such an imbalance leads to search spaces of different sizes across different stages and search times that may be dominated by those in a few stages.

To achieve a better balance of activities across stages, SGPlan\textsubscript{t}(ASPEN,N,DYN\textsubscript{P}) adjusts the boundary of stages dynamically. This is accomplished by finding \(M\), the number of time points in the horizon related to conflicts, at the end of the outer loop (Line 15) and by partitioning the horizon into \(N\) stages in such a way that each stage contains approximately the same number \((M/N)\) of such time points (Line 19). To determine the best \(N\), Figure 11 plots the number of iterations taken by static and dynamic partitioning in finding a feasible schedule of the 8-orbit CX1-PREF problem. The results show that \(N = 100\) is a good choice. Since other benchmarks lead to similar conclusions, we set \(N = 100\) in our experiments. Note that although \(N\) is relatively large, some stages will have all their local constraints satisfied as planning progresses. To avoid managing such defunct stages, our implementation collapses automatically adjacent defunct stages in such a way that each resulting stage contains at least one unsatisfied local constraint. Consequently, the actual number of stages used during planning can be much smaller than the value of \(N\) shown here.

**Experimental results.** Figure 12 compares the performance of ASPEN, SGPlan\textsubscript{t}(ASPEN,100,STATIC\textsubscript{P}), SGPlan\textsubscript{t}(ASPEN,1,STATIC\textsubscript{P}) (a version of our planner without partitioning), and SGPlan\textsubscript{t}(ASPEN,100,DYN\textsubscript{P}) on the four benchmarks described earlier in this section. In each graph, we plot the quality of the best feasible schedule found with respect to the number of search iterations. Although SGPlan\textsubscript{t} is not guaranteed to find optimal schedules, it can generate multiple locally optimal feasible schedules and keep improving on the best schedule found. In our experiments, we maintain the best schedule found as more search time is spent. The results show that descents using annealing in SGPlan\textsubscript{t}(ASPEN,1,STATIC\textsubscript{P}) have little improvements over the original ASPEN: they lead to better solutions in PREF but worse solutions in CX1-PREF with 16 orbits, DCAPS, and OPTIMIZE. Our results also show that SGPlan\textsubscript{t}(ASPEN,100,STATIC\textsubscript{P}) is able to find schedules of the same quality one to two orders of magnitude faster than ASPEN and SGPlan\textsubscript{t}(ASPEN,1,STATIC\textsubscript{P}), as well as much better schedules when they converge. Further, dynamic partitioning can lead to better schedules in shorter times than those of static partitioning. Hence, we conclude that improvements in SGPlan\textsubscript{t} are mainly due to partitioning and not to annealing.

*Artificial Intelligence* (accepted to appear)
In this section, we describe our results on partitioning PDDL2.1 benchmarks along their temporal horizons and on using the mixed-space MIPS planner [12] to solve the partitioned subproblems.

Figure 12: Quality-time comparisons of SGPlan\textsubscript{t}(ASPEN,1,STATIC\textsubscript{p}), SGPlan\textsubscript{t}(ASPEN,100,STATIC\textsubscript{p}), ASPEN, and SGPlan\textsubscript{t}(ASPEN,100,DYN\textsubscript{p}). (All runs involving SGPlan\textsubscript{t} were terminated at 24,000 iterations.)

5.2 SGPlan\textsubscript{t}(MIPS): A Planner Using MIPS to solve Partitioned Problems

In this section, we describe our results on partitioning PDDL2.1 benchmarks along their temporal horizons and on using the mixed-space MIPS planner [12] to solve the partitioned subproblems.
MIPS [12] is a heuristic planner that performs static analysis of a problem instance in mixed space and continuous time, searches for an optimized sequential plan, and performs a critical path analysis called PERT to generate optimal parallel plans from a sequence of operators and their precedence relations. Using a weighted A* algorithm, it finds an optimal feasible path from initial state $s_f$ to goal state $s_g \in G$ in a state space of propositional facts and numeric variables. It can also optimize an arbitrary objective by incorporating the objective in its heuristic function.

By generating approximate relaxed plans for each encountered state, MIPS uses the relaxed planning heuristic (RPH) [18] for guidance. RPH builds a relaxed plan by using the well-known planning graph proposed in Graphplan [5] but by ignoring the delete effects of actions. It then extracts a relaxed plan from the planning graph and computes an estimated distance from the current state to the goal state. MIPS extends RPH with numeric information by using a combined propositional and numeric forward/backward approximation scheme. It can also integrate PERT scheduling in its heuristic estimate in order to favor states with a smaller parallel plan length.

MIPS can handle the STRIPS subset of the PDDL language and can cope with numeric quantities and durations in PDDL 2.1 (Level 2-3 in PDDL+) [14]. In PDDL2.1, actions are represented by parameters, durations, conditions, and effects. A condition may be defined in terms of logical or functional expressions of ground atoms, and a conditional effect can be evaluated either at the start, the end, or during the interval of an action. MIPS can also handle some additional features from ADL, namely, negative preconditions and (universal) conditional effects.

MIPS competed in the second and the third International Planning Competitions and was awarded “Distinguished Performance” in the fully automated track in both. We use MIPS in our experiments because it performs well and its source code is readily available.

**Implementation of the partition-and-resolve search.** Figure 13 shows the pseudo code of SGPlan\(_t\)(MIPS,N). It generates an initial (possibly infeasible) plan of a planning problem, formulates the problem in a penalty function, decomposes the states into $N+1$ stages, solves each subproblem independently, and resolves the violated global constraints by increasing their penalties.

MIPS specifies the state of a problem as $s = (s_f, s_r)$, where $s_f$ contains the set of $n_f$ true facts at $s$, and $s_r$ is an $n_r$-vector of instantiated values of the numeric variables at $s$. It further partitions the set of grounded facts into symmetry groups [12] in the static-analysis phase in such a way that each element of $s_f$ is a fact from a unique symmetry group. For example, a small problem may have three symmetry groups:

- Group 1 = $(\text{at person}_1 \text{ city}_0, \text{at person}_1 \text{ city}_1, \text{at person}_1 \text{ city}_2, \text{in person}_1 \text{ plane}_1)$;
- Group 2 = $(\text{at person}_2 \text{ city}_0, \text{at person}_2 \text{ city}_1, \text{at person}_2 \text{ city}_2, \text{in person}_2 \text{ plane}_1)$;
- Group 3 = $(\text{at plane}_1 \text{ city}_0, \text{at plane}_1 \text{ city}_1, \text{at plane}_1 \text{ city}_2)$.

A valid state can have $s_f = (\text{at person}_1 \text{ city}_0, \text{at person}_2 \text{ city}_1, \text{at plane}_1 \text{ city}_2)$.

Based on the definition of symmetry groups, we define the neighborhood $N_m(s)$ of $s$ to include all states $s'$ that differ from $s$ by at most one fact, where the facts that differ between $s$ and $s'$ are in the same symmetry group. That is, $s$ and $s'$ are neighboring states when $s$ and $s'$ differ by only two facts $f \in s$ and $f' \in s'$ in the same symmetry group.

To generate a neighboring state $s'$ from $s$, we randomly pick a fact in $s$ and perturb it to a different fact in the same symmetry group. Note that, since $s_r$, the numeric part of $s$, is not changed in the process, there may not exist an action for a valid transition from $s$ to $s'$.

To quantify the notion of a valid transition, we measure the distance $D(s, q)$ between two states $s = (s_f, s_r)$ and $q = (q_f, q_r)$ as the number of different facts between $s$ and $q$ plus the normalized

Artificial Intelligence (accepted to appear)
1. procedure SGPlanₜ(MIPS,N)
2. generate initial plan using relaxed operators;
3. repeat
4. \( \text{iter} \leftarrow 0; \)
5. for \( t = 0 \) to \( N \) // initial state in Stage \( t \) is \( sᵢ(t) \) from initial plan //
6. \( \text{num}_{\text{trials}} \leftarrow 0; \)
7. repeat
8. \( \text{num}_{\text{trials}} \leftarrow \text{num}_{\text{trials}} + 1; \)
9. generate a new initial state in \( N_m(sᵢ(t)) \) for Stage \( t \);
10. call MIPS to solve \( P_t^{(i)} \) in Stage \( t \);
11. evaluate \( J^{(i)}(z) \) in (28) of the solution plan generated by MIPS;
12. until \( J^{(i)}(z) \) is improved or \( \text{num}_{\text{trials}} > \text{max}_{\text{trials}} \);
13. end_for
14. update penalty vector \( γ \) on violated global constraints;
15. \( \text{iter} \leftarrow \text{iter} + 1; \)
16. if \( (\text{iter} \mod τ) = 0 \) then repartition the stages end_if;
17. until no change in \( z \) and \( γ \) in an iteration;
18. return the best plan found;
19. end_procedure

Figure 13: SGPlanₜ(MIPS,N): The partition-and-resolve procedure in SGPlan that partitions a PDDL2.1 planning problem along its temporal horizon into \( N+1 \) subproblems, that calls MIPS to solve the subproblems, and that resolves the violated global constraints.

difference between their numerical parts if \( s \neq q \):

\[
D(s,q) = (\text{number of different facts between } s \text{ and } q) + \sum_{i=1}^{n_r} \frac{|s_{r_i} - q_{r_i}|}{\max(s_{r_i}, q_{r_i})} |_{s \neq q}.
\]  

Hence, \( D(s,q) = 0 \) if and only if \( s \) and \( q \) are identical states. We further define \( S(s) \), the set of successor (different from neighborhood) states of \( s \), in such a way that there exists a valid action that brings \( s \) to \( q \) for all \( q \in S(s) \). Last, we define the transition distance \( T(s,q) \) to be the minimum distance between \( s \) and \( q \) over all successors of \( s \):

\[
T(s,q) = \min_{v \in S(s)} D(v,q).
\]  

According to this definition, \( T(s,q) = 0 \) when there exists a valid action to bring \( s \) to \( q \).

Based on the concepts on neighborhood, state transition, and transition distance, we can now specify the local planning subproblem \( P_t^{(i)} \) in Stage \( t \). After partitioning, \( P_t^{(i)} \) has initial state \( sᵢ(t) \) and goal state \( sᵢ(t+1) \). (See Figure 14 for the states defined in Stage \( t \).) Since this initial local plan may be infeasible, we need to formulate \( P_t^{(i)} \) that, when solved, will hopefully make the overall planning problem feasible (Line 10 of Figure 13). This subproblem has the same domain specification as the original problem, initial state \( sᵢ(t) \in N_m(sᵢ(t)) \), and goal state \( sᵢ(t+1) \). In addition, there are two global constraints at the boundaries between Stage \( t \) and the predecessor and successor stages:

\[
H_{t-1}(z) = T(sᵢ(t-1), sᵢ'(t)) = 0; \quad H_t(z) = T(sᵢ(t), sᵢ(t+1)) = 0. \]  

Hence, \( H_{t-1}(z) = 0 \) (resp. \( H_t(z) = 0 \)) is satisfied if and only if there is a valid action to bring \( sᵢ(t-1) \) (resp. \( sᵢ(t) \)) to \( sᵢ(t) \) (resp. \( sᵢ(t+1) \)). These global constraints are then added as biases in the objective of \( P_t^{(i)} \) as follows:

\[
J^{(i)}(z) = J(z) + γ_{t-1}H_{t-1}(z) + γ_tH_t(z), \]  

Artificial Intelligence (accepted to appear)
where \( \gamma_{t-1} \) and \( \gamma_t \) are the fixed penalties associated with the two global constraints when \( P_t^{(t)} \) is solved. The other constraints of the subproblem in Stage \( t \) remain unchanged.

After solving \( P_t^{(t)} \), MIPS returns a locally optimal feasible plan from \( s'_i(t) \) to \( s_i(t+1) \) if one exists; otherwise, it returns a feasible plan from \( s'_i(t) \) to \( s_g(t) \) that minimizes (32). We accept this plan if it improves \( J(t)(z) \); otherwise, we repeat the process by using a new initial state in \( \mathcal{N}_m(s_i(t)) \) until we find a better plan, or when the maximum number of trials is exceeded (Line 12 of Figure 13). In our experiments, we set max trials to 5.

After completing the \( N+1 \) subproblems in an iteration (Line 14), we update the penalties of all violated global constraints, using \( \omega > 0 \) to control the rate of increase:

\[
\gamma_t \leftarrow \gamma_t + \omega \cdot H_t(z), \quad t = 0, 1, \ldots, N.
\] (33)

We set heuristically \( \omega = 0.01 J_a \), where \( J_a \) is the average value of \( J(z) \) in the last three iterations.

Similar to the partition-and-resolve implementation of ASPEN, we repartition the stages dynamically by adjusting the boundary of stages every certain number \( \tau \) in Figure 13) of iterations. This is accomplished by counting the number of state transitions from \( s_i \) to \( s_G \) at the end of the outer loop (Line 16) and by redefining the stage boundaries in order for each stage to have approximately the same number of state transitions. After repartitioning, the number of violated global constraints in a stage may be different from one. In our experiments, we set \( N = 20 \) and \( \tau = 5 \).

Experimental results. We show that SGPlan4(MIPS,20) improves significantly over the original MIPS planner on a set of PDDL2.1 planning benchmarks used in the Third International Planning Competition. The problems studied belong to a number of domains, including DepotNumeric, DepotSim, DepotTime, DriveLogNumeric, DriveLogSim, DriveLogTime, ZenoTravelNumeric, ZenoTravelSim, and ZenoTravelTime.

As a reference, we also show the performance of LPG, the best automated planner in the competition. Because we did not have access to the source code of LPG at the time of our experiments, we were not able to report the performance of using LPG as a basic solver in SGPlan4.

We conducted our experiments on an AMD Athlon MP2000 PC with Redhat Linux 7.2. In our experiments on MIPS and LPG, we used the August-2003 version of their executables with default

---

4At the time of this revision, we have finished implementing SGPlan4(FF/LPG,N), a planner that partitions the constraints of a problem instance according to its subgoals into \( N+1 \) subproblems and that calls either FF or LPG to solve the subproblems [7]. SGPlan4(FF/LPG,N) participated in the Fourth International Planning Competition and won the first prize in the suboptimal temporal metric track and the second prize in the suboptimal propositional track. It ranked better than a new version of LPG in both tracks. Due to the extensive redesign involved, we plan to report its features and performance in a future paper.

Artificial Intelligence (accepted to appear)
5.2 SGPlan\textsubscript{t}(MIPS): A Planner Using MIPS to solve Partitioned Problems

![Graphs showing normalized quality and time comparisons](image)

Figure 15: Normalized time and quality of SGPlan\textsubscript{t}(MIPS,20) with respect to MIPS on the 120 instances solvable by MIPS (out of a total of 160 instances). The time and quality of MIPS are normalized to (1,1).

parameters downloaded from their Web sites. In accordance to the way that planners were run in the International Planning Competitions, we used a fixed random seed of 1000 in LPG, MIPS, and SGPlan\textsubscript{t}(MIPS,20), making them behave like deterministic planners. We also used the same parameters specified for the original MIPS in the version of MIPS embedded in SGPlan\textsubscript{t}(MIPS,20). We then ran each planner once on each problem instance for a maximum time limit of 1,000 sec.

For the 120 (out of a total of 160) instances solvable by MIPS, Figure 15a compares the quality of the solution of each instance found by MIPS and by SGPlan\textsubscript{t}(MIPS,20) when it was given the same amount of time taken by MIPS to solve that instance. It measures the fraction of instances that SGPlan\textsubscript{t}(MIPS,20) found a better solution using the same amount of time taken by MIPS. In contrast, Figure 15b compares the time taken by MIPS to solve an instance and that by SGPlan\textsubscript{t}(MIPS,20) when it found a solution of the same or better quality as that of MIPS for that instance. It measures the fraction of instances that SGPlan\textsubscript{t}(MIPS,20) found a solution faster and of the same or better quality as that of MIPS. The graphs do not include the results on the 30 instances for which SGPlan\textsubscript{t}(MIPS,20) could solve but MIPS could not find any feasible plan in 1,000 sec. The results show that SGPlan\textsubscript{t}(MIPS,20) is able to improve over MIPS in 81.7% of the cases in quality or 83.2% of the cases in time on the PDDL2.1 instances solvable by MIPS. In comparison, an implementation of SGPlan\textsubscript{g}(MIPS) that partitions a problem by its subgoals leads to comparable performance and is able to improve over MIPS in 80.5% of the cases in quality or 80.1% of the cases in time on the PDDL2.1 instances solvable by MIPS [48].

Of the 150 of the 160 instances solvable by SGPlan\textsubscript{t}(MIPS,20), SGPlan\textsubscript{t}(MIPS,20) can find a feasible solution with better time (resp. quality) than MIPS in 94.4% (resp. 93.8%).

As a reference, Figure 16 shows that SGPlan\textsubscript{t}(MIPS,20) has comparable normalized performance with respect to that of LPG. The results show that SGPlan\textsubscript{t}(MIPS,20) is able to improve over LPG in 49.6% of the cases in quality or 54.2% of the cases in time. The improvements over LPG are not substantial because SGPlan\textsubscript{t}(MIPS,20) inherits MIPS’ limitations in its performance and may not be able to improve over LPG when MIPS performs worse than LPG to start with.

Table 1 presents the complete results on the 160 instances tested. Since MIPS was not designed to work in an anytime mode and can find only one solution, whereas SGPlan\textsubscript{t}(MIPS,20) and LPG can generate multiple solutions with improving quality, we list for each instance the solution time and quality of MIPS, and those of the first and the final solutions found by SGPlan\textsubscript{t}(MIPS,20) and LPG. The results show that SGPlan\textsubscript{t}(MIPS,20) outperforms MIPS in most of the instances tested, and that SGPlan\textsubscript{t}(MIPS,20) has comparable performance as LPG.

*Artificial Intelligence* (accepted to appear)
6 Conclusions

In this paper, we have presented a new theory of penalty methods for continuous, discrete, and mixed-integer optimization and its application in solving temporal planning problems partitioned by constraints. Our theory shows that a constrained local minimum of a general MINLP problem has a one-to-one correspondence to an extended saddle point of a penalty function with non-negative (transformed) constraint functions, when its penalties are larger than some thresholds. Hence, one way to find a constrained local minimum of the MINLP is to increase gradually the penalties of those violated constraints and to look for a local minimum of the penalty function using any existing algorithm until a solution to the constrained model is found. Next, by defining a proper neighborhood for MINLPs, we show the extension of the method to constraint-partitioned MINLPs. Finally, by partitioning along the time horizon and by using the discrete-space ASPEN and the mixed-space MIPS planners to solve partitioned planning subproblems, we have demonstrated significant improvements on some benchmark problems, both in terms of the quality of the plans generated and the execution times to find them. Results on partitioning planning problems along the subgoal dimension using the MIPS planner have been reported elsewhere [47, 48].

Our constraint-partitioning approach is important for reducing the complexity of nonlinear constrained planning problems. It leads to subproblems that are much easier to solve because each has a significantly smaller number of constraints. It also results in subproblems that are very similar to the original problem and, therefore, can be evaluated by existing planners with little or no modification. Partitioning, however, introduces violated global constraints that may have to be resolved after solving the subproblems. To reduce the amount of backtracking in resolving such global constraints, we have developed new necessary conditions that are much stronger than the local constraints for limiting the search space to be backtracked in each subproblem.

The results presented can be generalized to solve nonlinear constrained optimization problems in many engineering applications [49]. Our new theory will allow nonlinear problems in continuous, discrete, and mixed spaces to be solved in a unified and efficient fashion.
Table 1: Results on MIPS, SGPlan\(_{(MIPS,20)}\) and LPG in solving some PDDL2.1 benchmark instances. All timing results are in milliseconds, and all solvers were run with a maximum time limit of 1,000 sec. “—” means that no solution was found in the time limit. For MIPS, Time and Sol list the solution time and quality (lower are better). For SGPlan\(_{(MIPS,20)}\) and LPG, Time\(_{1}\) and Sol\(_{1}\) list the time and quality of the first solution found, and Time\(_{f}\) and Sol\(_{f}\) list the time and quality of the last solution found in the time limit. For each instance, a boxed number indicates the best quality between MIPS and SGPlan\(_{(MIPS,20)}\) on the last solution found in the time limit, whereas the result of LPG is underlined when LPG has better quality than SGPlan\(_{(MIPS,20)}\) (irrespective of time) on the last solution found in the time limit, or when they have the same final quality and LPG requires a smaller CPU time.

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>MIPS</th>
<th>SGPlan(_{(MIPS,20)})</th>
<th>LPG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Sol</td>
<td>Time(_{f})</td>
</tr>
<tr>
<td>DepotNumeric1</td>
<td>149 32</td>
<td>20</td>
<td>22.6</td>
</tr>
<tr>
<td>DepotNumeric2</td>
<td>398 43</td>
<td>40</td>
<td>37.4</td>
</tr>
<tr>
<td>DepotNumeric3</td>
<td>2543 31</td>
<td>100</td>
<td>23</td>
</tr>
<tr>
<td>DepotNumeric4</td>
<td>– –</td>
<td>1930</td>
<td>54.6</td>
</tr>
<tr>
<td>DepotNumeric5</td>
<td>– –</td>
<td>30340</td>
<td>129.5</td>
</tr>
<tr>
<td>Depot Numeric6</td>
<td>– –</td>
<td>– –</td>
<td>– –</td>
</tr>
<tr>
<td>Depot Numeric7</td>
<td>1522 38</td>
<td>160</td>
<td>96.1</td>
</tr>
<tr>
<td>Depot Numeric8</td>
<td>1605 28</td>
<td>2180</td>
<td>35</td>
</tr>
<tr>
<td>Depot Numeric9</td>
<td>– –</td>
<td>– –</td>
<td>– –</td>
</tr>
<tr>
<td>Depot Numeric10</td>
<td>107850 16</td>
<td>140</td>
<td>15</td>
</tr>
<tr>
<td>DepotSim1</td>
<td>38</td>
<td>46.12</td>
<td>30</td>
</tr>
<tr>
<td>DepotSim2</td>
<td>51</td>
<td>73.2</td>
<td>30</td>
</tr>
<tr>
<td>DepotSim3</td>
<td>476</td>
<td>103.39</td>
<td>620</td>
</tr>
<tr>
<td>DepotSim4</td>
<td>135485</td>
<td>130.03</td>
<td>290</td>
</tr>
<tr>
<td>DepotSim5</td>
<td>– –</td>
<td>24400</td>
<td>157</td>
</tr>
<tr>
<td>DepotSim6</td>
<td>– –</td>
<td>– –</td>
<td>– –</td>
</tr>
<tr>
<td>DepotSim7</td>
<td>169</td>
<td>67.27</td>
<td>1080</td>
</tr>
<tr>
<td>DepotSim8</td>
<td>302327</td>
<td>111.039</td>
<td>530</td>
</tr>
<tr>
<td>DepotSim9</td>
<td>– –</td>
<td>39680</td>
<td>254</td>
</tr>
<tr>
<td>DepotSim10</td>
<td>164389</td>
<td>91.0 4</td>
<td>130</td>
</tr>
<tr>
<td>DepotSim11</td>
<td>– –</td>
<td>2780</td>
<td>194</td>
</tr>
<tr>
<td>DepotSim12</td>
<td>– –</td>
<td>– –</td>
<td>– –</td>
</tr>
<tr>
<td>DepotSim13</td>
<td>450</td>
<td>84.025</td>
<td>350</td>
</tr>
<tr>
<td>DepotSim14</td>
<td>– –</td>
<td>470</td>
<td>56</td>
</tr>
<tr>
<td>DepotSim15</td>
<td>– –</td>
<td>159490</td>
<td>262</td>
</tr>
<tr>
<td>DepotSim16</td>
<td>439830</td>
<td>73.032</td>
<td>200</td>
</tr>
<tr>
<td>DepotTime1</td>
<td>40</td>
<td>59.4811</td>
<td>40</td>
</tr>
<tr>
<td>DepotTime2</td>
<td>54</td>
<td>92.3111</td>
<td>40</td>
</tr>
<tr>
<td>DepotTime3</td>
<td>623</td>
<td>255.082</td>
<td>2120</td>
</tr>
<tr>
<td>DepotTime4</td>
<td>143242</td>
<td>199.456</td>
<td>180</td>
</tr>
<tr>
<td>DepotTime5</td>
<td>– –</td>
<td>58520</td>
<td>556.905</td>
</tr>
<tr>
<td>DepotTime6</td>
<td>– –</td>
<td>– –</td>
<td>– –</td>
</tr>
<tr>
<td>DepotTime7</td>
<td>134</td>
<td>149.791</td>
<td>280</td>
</tr>
<tr>
<td>DepotTime8</td>
<td>754</td>
<td>136.313</td>
<td>3300</td>
</tr>
<tr>
<td>DepotTime9</td>
<td>– –</td>
<td>11371</td>
<td>1083.33</td>
</tr>
<tr>
<td>DepotTime10</td>
<td>172322</td>
<td>245.57</td>
<td>130</td>
</tr>
<tr>
<td>DepotTime11</td>
<td>– –</td>
<td>31090</td>
<td>535.334</td>
</tr>
<tr>
<td>DepotTime12</td>
<td>– –</td>
<td>334500</td>
<td>194.096</td>
</tr>
<tr>
<td>DepotTime13</td>
<td>1203</td>
<td>104.526</td>
<td>430</td>
</tr>
<tr>
<td>DepotTime14</td>
<td>– –</td>
<td>870</td>
<td>370.199</td>
</tr>
<tr>
<td>DepotTime15</td>
<td>– –</td>
<td>242110</td>
<td>268.596</td>
</tr>
<tr>
<td>DepotTime16</td>
<td>429824</td>
<td>84.8653</td>
<td>230</td>
</tr>
<tr>
<td>DepotTime17</td>
<td>– –</td>
<td>1720</td>
<td>97.05</td>
</tr>
</tbody>
</table>

*continued on next page*

Artificial Intelligence (accepted to appear)
## 6 CONCLUSIONS

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>MIPS</th>
<th>SOFPlan(MIPS,20)</th>
<th>LFG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Sol</td>
<td>Time</td>
</tr>
<tr>
<td>DriveLogNumeric1</td>
<td>90</td>
<td>1099 23.2</td>
<td>42920</td>
</tr>
<tr>
<td>DriveLogNumeric2</td>
<td>89</td>
<td>1497 23.0</td>
<td>324250</td>
</tr>
<tr>
<td>DriveLogNumeric3</td>
<td>92</td>
<td>907 23.6</td>
<td>6240</td>
</tr>
<tr>
<td>DriveLogNumeric4</td>
<td>112</td>
<td>715 23.9</td>
<td>5490</td>
</tr>
<tr>
<td>DriveLogNumeric5</td>
<td>124</td>
<td>878 23.5</td>
<td>247570</td>
</tr>
<tr>
<td>DriveLogNumeric6</td>
<td>130</td>
<td>1467 23.0</td>
<td>52150</td>
</tr>
<tr>
<td>DriveLogNumeric7</td>
<td>123</td>
<td>866 23.0</td>
<td>131630</td>
</tr>
<tr>
<td>DriveLogNumeric8</td>
<td>196802</td>
<td>3273 23.0</td>
<td>31380</td>
</tr>
<tr>
<td>DriveLogNumeric9</td>
<td>629</td>
<td>3002 23.6</td>
<td>38640</td>
</tr>
<tr>
<td>DriveLogNumeric10</td>
<td>278</td>
<td>402 23.6</td>
<td>346800</td>
</tr>
<tr>
<td>DriveLogNumeric11</td>
<td>7250</td>
<td>616 23.6</td>
<td>151870</td>
</tr>
<tr>
<td>DriveLogNumeric12</td>
<td>14320 23.0</td>
<td>3227 23.0</td>
<td>49726</td>
</tr>
<tr>
<td>DriveLogNumeric13</td>
<td>2521 23.4</td>
<td>2148 23.0</td>
<td>500690</td>
</tr>
<tr>
<td>DriveLogNumeric14</td>
<td>344331.1 23.6</td>
<td>3347 23.0</td>
<td>10692.5</td>
</tr>
<tr>
<td>DriveLogNumeric15</td>
<td>12421 23.0</td>
<td>1753 23.0</td>
<td>489580</td>
</tr>
<tr>
<td>DriveLogNumeric16</td>
<td>74893 23.0</td>
<td>43983 23.0</td>
<td>74893</td>
</tr>
<tr>
<td>DriveLogNumeric17</td>
<td>19513 23.0</td>
<td>20583.4 23.0</td>
<td>584000</td>
</tr>
<tr>
<td>DriveLogNumeric18</td>
<td>45051 23.0</td>
<td>12480.1 23.0</td>
<td>250850</td>
</tr>
<tr>
<td>DriveLogNumeric19</td>
<td>605000 23.0</td>
<td>25219.6 23.0</td>
<td>605000</td>
</tr>
<tr>
<td>DriveLogNumeric20</td>
<td>484120 23.0</td>
<td>19323.2 23.0</td>
<td>564590</td>
</tr>
<tr>
<td>DriveLogSim1</td>
<td>90</td>
<td>92.07 23.0</td>
<td>920000</td>
</tr>
<tr>
<td>DriveLogSim2</td>
<td>90</td>
<td>92.21 23.0</td>
<td>10330</td>
</tr>
<tr>
<td>DriveLogSim3</td>
<td>98</td>
<td>[40.07] 23.0</td>
<td>4700</td>
</tr>
<tr>
<td>DriveLogSim4</td>
<td>99</td>
<td>89.16 23.0</td>
<td>9800</td>
</tr>
<tr>
<td>DriveLogSim5</td>
<td>112</td>
<td>51.19 23.0</td>
<td>119704</td>
</tr>
<tr>
<td>DriveLogSim6</td>
<td>117</td>
<td>64.13 23.0</td>
<td>94030</td>
</tr>
<tr>
<td>DriveLogSim7</td>
<td>122</td>
<td>40.09 23.0</td>
<td>5100</td>
</tr>
<tr>
<td>DriveLogSim8</td>
<td>279.1</td>
<td>111.26 23.0</td>
<td>13400</td>
</tr>
<tr>
<td>DriveLogSim9</td>
<td>202</td>
<td>264.31 23.0</td>
<td>19200</td>
</tr>
<tr>
<td>DriveLogSim10</td>
<td>269.1</td>
<td>61.21 23.0</td>
<td>50680</td>
</tr>
<tr>
<td>DriveLogSim11</td>
<td>351</td>
<td>99.21 23.0</td>
<td>8500</td>
</tr>
<tr>
<td>DriveLogSim12</td>
<td>1772 23.0</td>
<td>25241.2 23.0</td>
<td>442350</td>
</tr>
<tr>
<td>DriveLogSim13</td>
<td>1734 23.0</td>
<td>104.29 23.0</td>
<td>258250</td>
</tr>
<tr>
<td>DriveLogSim14</td>
<td>2403 23.0</td>
<td>226.44 23.0</td>
<td>156350</td>
</tr>
<tr>
<td>DriveLogSim15</td>
<td>13620 23.0</td>
<td>265.43 23.0</td>
<td>700311</td>
</tr>
<tr>
<td>DriveLogSim16</td>
<td>202</td>
<td>264.31 23.0</td>
<td>19200</td>
</tr>
<tr>
<td>DriveLogSim17</td>
<td>1750 23.0</td>
<td>224.94 23.0</td>
<td>43500</td>
</tr>
<tr>
<td>DriveLogSim18</td>
<td>659200 23.0</td>
<td>67200 23.0</td>
<td>208150</td>
</tr>
<tr>
<td>DriveLogSim19</td>
<td>580800 23.0</td>
<td>10300 23.0</td>
<td>580800</td>
</tr>
<tr>
<td>DriveLogSim20</td>
<td>178550 23.0</td>
<td>77100 23.0</td>
<td>311250</td>
</tr>
</tbody>
</table>

Artificial Intelligence (accepted to appear)
<table>
<thead>
<tr>
<th>Problem ID</th>
<th>MIPS</th>
<th>SOGPlan$_{\text{MIPS}, 20}$</th>
<th>LFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>DriveLogTime9</td>
<td>233</td>
<td>980</td>
<td>40</td>
</tr>
<tr>
<td>DriveLogTime10</td>
<td>287</td>
<td>340</td>
<td>50</td>
</tr>
<tr>
<td>DriveLogTime11</td>
<td>343</td>
<td>391</td>
<td>50</td>
</tr>
<tr>
<td>DriveLogTime12</td>
<td>1530</td>
<td>611</td>
<td>260</td>
</tr>
<tr>
<td>DriveLogTime13</td>
<td>1256</td>
<td>558</td>
<td>120</td>
</tr>
<tr>
<td>DriveLogTime14</td>
<td>2303</td>
<td>1049</td>
<td>260</td>
</tr>
<tr>
<td>DriveLogTime15</td>
<td>9853</td>
<td>893</td>
<td>970</td>
</tr>
<tr>
<td>DriveLogTime16</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>DriveLogTime17</td>
<td>236244</td>
<td>954.94</td>
<td>3590</td>
</tr>
<tr>
<td>DriveLogTime18</td>
<td>--</td>
<td>--</td>
<td>32470</td>
</tr>
<tr>
<td>DriveLogTime19</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>DriveLogTime20</td>
<td>--</td>
<td>--</td>
<td>94730</td>
</tr>
<tr>
<td>ZenoTravelNumeric1</td>
<td>72</td>
<td>13564</td>
<td>16</td>
</tr>
<tr>
<td>ZenoTravelNumeric2</td>
<td>70</td>
<td>12786</td>
<td>16</td>
</tr>
<tr>
<td>ZenoTravelNumeric3</td>
<td>92</td>
<td>7505</td>
<td>30</td>
</tr>
<tr>
<td>ZenoTravelNumeric4</td>
<td>91</td>
<td>16699</td>
<td>30</td>
</tr>
<tr>
<td>ZenoTravelNumeric5</td>
<td>100</td>
<td>19916</td>
<td>30</td>
</tr>
<tr>
<td>ZenoTravelNumeric6</td>
<td>112</td>
<td>35282</td>
<td>40</td>
</tr>
<tr>
<td>ZenoTravelNumeric7</td>
<td>103.1</td>
<td>16472</td>
<td>30</td>
</tr>
<tr>
<td>ZenoTravelNumeric8</td>
<td>183</td>
<td>33543</td>
<td>50</td>
</tr>
<tr>
<td>ZenoTravelNumeric9</td>
<td>192</td>
<td>28047</td>
<td>70</td>
</tr>
<tr>
<td>ZenoTravelNumeric10</td>
<td>214</td>
<td>79664</td>
<td>70</td>
</tr>
<tr>
<td>ZenoTravelNumeric11</td>
<td>252</td>
<td>55480</td>
<td>250</td>
</tr>
<tr>
<td>ZenoTravelNumeric12</td>
<td>306</td>
<td>41310</td>
<td>120</td>
</tr>
<tr>
<td>ZenoTravelNumeric13</td>
<td>413</td>
<td>82230</td>
<td>140</td>
</tr>
<tr>
<td>ZenoTravelNumeric14</td>
<td>6247</td>
<td>233381</td>
<td>120</td>
</tr>
<tr>
<td>ZenoTravelNumeric15</td>
<td>15890</td>
<td>147618</td>
<td>3090</td>
</tr>
<tr>
<td>ZenoTravelNumeric16</td>
<td>31652</td>
<td>143282</td>
<td>12920</td>
</tr>
<tr>
<td>ZenoTravelNumeric17</td>
<td>64438</td>
<td>182558</td>
<td>302440</td>
</tr>
<tr>
<td>ZenoTravelNumeric18</td>
<td>123543.4</td>
<td>70794</td>
<td>498710</td>
</tr>
<tr>
<td>ZenoTravelNumeric19</td>
<td>139395</td>
<td>212997</td>
<td>594600</td>
</tr>
<tr>
<td>ZenoTravelNumeric20</td>
<td>245335</td>
<td>89937</td>
<td>1054220</td>
</tr>
</tbody>
</table>

Artificial Intelligence (continued to appear)
Appendix A: Proof of Theorem 1

The proof consists of two parts.

“⇒” part: Given $x^*$, we need to prove that there exist finite $\alpha^{**} > \alpha^* \geq 0$ and $\beta^{**} > \beta^* \geq 0$ that satisfy (13). The first inequality in (13) is true for all $\alpha$ and $\beta$ because $x^*$ is a CLM$_c$, which implies $|h(x^*)| = 0$ and max$(0, g(x^*)) = 0$.

To prove the second inequality in (13), we prove for any $x \in \mathcal{N}_c(x^*)$ that there exist finite $\alpha^*$ and $\beta^*$ such that the inequality is satisfied for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. Let $x = x^* + \epsilon \overrightarrow{p}$, where $\|\overrightarrow{p}\| = 1$ is a unit directional vector and $\epsilon$ is an infinitely small positive scalar. We consider the following four cases.

1) If all the constraints are inactive inequality constraints, then $x \in \mathcal{N}_c(x^*)$ is also a feasible point. Hence, (13) implies $f(x) \geq f(x^*)$ and, regardless the choice of the penalties,

$$L_c(x, \alpha^{**}, \beta^{**}) = f(x) \geq f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}).$$

(34)

2) If there exists an equality constraint function $h_k$ that is discontinuous along $\overrightarrow{p}$, then for a small enough $\epsilon$, there exists a finite positive $\xi$ such that:

$$|h_k(x)| > \xi > 0 = h_k(x^*).$$

(35)

The above must be true because $h_k(x)$ would be continuous along $\overrightarrow{p}$ if (35) were false.

If we set $\alpha^{**}_k > \alpha^*_k = 1$ and when $\epsilon$ is small enough, then from (35):

$$L_c(x, \alpha^{**}, \beta^{**}) = f(x) + \sum_{i=1}^{m} \alpha^{**}_i |h_i(x)| + \sum_{j=1}^{r} \beta^{**}_j \max(0, g_j(x))$$

$$\geq f(x) + \alpha^{**}_k |h_k(x)| > f(x^*) + \epsilon \nabla_x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \alpha^*_k \xi$$

$$> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}).$$

(36)

3) If there exists an active inequality constraint function $g_k$ that is discontinuous along $\overrightarrow{p}$, then for a small enough $\epsilon$, there exists a finite positive $\xi$ such that max$(0, g_k(x)) > \xi > 0$. If we set

Artificial Intelligence (accepted to appear)
\[ \beta_{k^*} > \beta_k^* = 1 \text{ and when } \epsilon \text{ is small enough, this condition implies that:} \]

\[ L_c(x, \alpha^{**}, \beta^{**}) = f(x) + \sum_{i=1}^{m} \alpha_i^{**} |h_i(x)| + \sum_{j=1}^{r} \beta_j^{**} \max(0, g_j(x)) \]

\[ \geq f(x) + \beta_k^{**} \max(0, g_k(x)) \geq f(x^*) + \epsilon \nabla x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \beta_k^* \xi > f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \] (37)

4) Other than inactive inequality constraints, if there are equality and active inequality constraint functions that are continuous along \( \overrightarrow{p} \), then according to the constraint-qualification condition, there must exist an equality or an active inequality constraint function that has non-zero subdifferential along \( \overrightarrow{p} \). Suppose there exists an equality constraint function \( h_k \) that has non-zero subdifferential along \( \overrightarrow{p} \) (the case with an active inequality constraint function is similar), which means \( |D_x(h_k(x^*), \overrightarrow{p})| > 0 \). If we set \( \alpha_k^{**} > \frac{|\nabla_x f(x^*)^T \overrightarrow{p}|}{|D_x(h_k(x^*), \overrightarrow{p})|} \) and when \( \epsilon \) is small enough, then:

\[ L_c(x, \alpha^{**}, \beta^{**}) = f(x) + \sum_{i=1}^{m} \alpha_i^{**} |h_i(x)| + \sum_{j=1}^{r} \beta_j^{**} \max(0, g_j(x)) \]

\[ \geq f(x) + \alpha_k^{**} |h_k(x)| \geq f(x^*) + \epsilon \nabla x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \alpha_k^* \epsilon |D_x(h_k(x^*), \overrightarrow{p})| \]

\[ \geq f(x^*) + \epsilon \left( \alpha_k^{**} |D_x(h_k(x^*), \overrightarrow{p})| - |\nabla x f(x^*)^T \overrightarrow{p}| \right) + o(\epsilon^2) \]

\[ > f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \] (38)

The second inequality in (13) is proved after combining Cases (1) to (4).

“\( \Rightarrow \)” part: Assuming (13) is satisfied, we need to prove that \( x^* \) is a CLM. Point \( x^* \) is feasible because the first inequality in (13) can only be satisfied when \( h(x^*) = 0 \) and \( g(x^*) \leq 0 \). Since \( |h(x^*)| = 0 \) and \( \max(0, g(x^*)) = 0 \), the second inequality in (13) ensures that \( x^* \) is a local minimum when compared to all feasible points in \( \mathcal{N}_c(x^*) \). Therefore, \( x^* \) is a CLM.

### Appendix B: Proof of Theorem 2

An earlier proof [55, 51] is rewritten in terms of our penalty formulation. It consists of two parts:

“\( \Leftarrow \)” part: Given \( y^* \), we need to prove that there exist finite \( \alpha^{**} > \alpha^* \geq 0 \) and \( \beta^{**} > \beta^* \geq 0 \) that satisfy (16). In order for \( \alpha^* \) and \( \beta^* \) to exist for every CLM, \( y^* \), \( \alpha^* \) and \( \beta^* \) must be bounded and be found in finite time. Given \( y^* \), consider all \( y \in \mathcal{N}_d(y^*) \), and let the initial \( \alpha^* = \beta^* = 0 \). For every \( y \) such that \( |h(y)| > 0 \) (resp. \( \max(0, g(y)) > 0 \)), there is at least one constraint that is not satisfied. For each such constraint, we update its penalty as follows:

\[ \alpha_i^* \leftarrow \max \left\{ \alpha_i^*, \frac{f(y^*) - f(y)}{|h_i(y)|} \right\} \quad \text{if } |h_i(y)| > 0 \] (39)

\[ \beta_j^* \leftarrow \max \left\{ \beta_j^*, \frac{f(y^*) - f(y)}{\max(0, g_j(y))} \right\} \quad \text{if } \max(0, g_j(y)) > 0. \] (40)

This update is repeated for every violated constraint of \( P_d \) and every \( y \in \mathcal{N}_d(y^*) \) until no further update is possible. The key of the proof is that, since \( \mathcal{N}_d(y^*) \) has a finite number of elements in discrete space, the update will terminate in finite time and result in finite \( \alpha^* \) and \( \beta^* \) values.

Next, we prove that the \( (y^*, \alpha^{**}, \beta^{**}) \) found satisfies (16). The proof of the first inequality in (16) is trivial because \( L_d(y^*, \alpha, \beta) = f(y^*) = L_d(y^*, \alpha^{**}, \beta^{**}) \).

Artificial Intelligence (accepted to appear)
For the second inequality in (16), since $y^*$ is a $CLM_d$, it is clear for all $y \in \mathcal{N}_d(y^*)$ where $h(y) = 0$ and $g(y) \leq 0$ that:

$$L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) \leq f(y) = L_d(y, \alpha^{**}, \beta^{**}).$$

(41)

For all $y \in \mathcal{N}_d(y^*)$ such that $h(y) \neq 0$ (resp. $g(y) \neq 0$), there must exist at least one constraint that is not satisfied. From (39) and (40), we know for this constraint that:

$$\sum_{i=1}^{m} \alpha^{**}_i > \alpha^{*}_i \geq \frac{f(y^*) - f(y)}{|h_i(y)|} \implies L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) < f(y) + \alpha^{**}_i|h_i(y)|$$

(42)

if $|h_i(y)| > 0,$

$$\beta^{**}_j > \beta^{*}_j \geq \frac{f(y^*) - f(y)}{\max(0, g_j(y))} \implies L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) < f(y) + \beta^{**}_j \max(0, g_j(y))$$

(43)

if $\max(0, g_j(y)) > 0.$

Further, since $\sum_{k=1, k \neq i}^{m} \alpha^{*}_k|h_k(y)| \geq 0$ (resp. $\sum_{r=1, k \neq j}^{r} \beta^{*}_k \max(0, g_j(y)) \geq 0$), it is clear that:

$$L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) \leq f(y) + \sum_{i=1}^{m} \alpha^{*}_i|h_i(y)| + \sum_{j=1}^{r} \beta^{*}_j \max(0, g_j(y)) = L_d(y, \alpha^{**}, \beta^{**}).$$

Hence, $(y^*, \alpha^{**}, \beta^{**})$ satisfies (16).

“$\Rightarrow$” part: Assuming (16) is satisfied, we need to prove that $y^*$ is a $CLM_d$. The proof is straightforward and is similar to that of Theorem 1.

Appendix C: Proof of Theorem 3

The proof consists of two parts.

“$\Rightarrow$” part: Given $(x^*, y^*)$, we need to prove that there exist finite $\alpha^{*} \geq 0$ and $\beta^{*} \geq 0$ so that $(x^*, y^*, \alpha^{**}, \beta^{**})$ satisfies (20). The first inequality in (20) is true for all $\alpha$ and $\beta$, since $(x^*, y^*)$ is a $CLM_m$ and $|h(x^*, y^*)| = \max(0, g(x^*, y^*)) = 0$.

To prove the second inequality in (20), we know that fixing $y$ at $y^*$ converts $P_m$ into $P_c$. Further, from Theorem 1, there exist finite $\alpha^{*}_c$ and $\beta^{*}_c$ such that:

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y^*, \alpha^{**}, \beta^{**}), \quad \forall x \in \mathcal{N}_c(x^*)|y^*, \quad \alpha^{**} > \alpha^{*}_c \geq 0, \quad \text{and} \quad \beta^{**} > \beta^{*}_c \geq 0.\tag{44}$$

Similarly, fixing $x$ at $x^*$ converts $P_m$ into $P_d$. Hence, from Theorem 2, we know that there exist finite $\alpha^{*}_d$ and $\beta^{*}_d$ such that, for the same $\alpha^{**}$ and $\beta^{**}$ in (44):

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x^*, y, \alpha^{**}, \beta^{**}), \quad \forall y \in \mathcal{N}_d(y^*)|x^*, \quad \alpha^{**} > \alpha^{*}_d \geq 0, \quad \text{and} \quad \beta^{**} > \beta^{*}_d \geq 0.\tag{45}$$

Since all $(x, y) \in \mathcal{N}_m(x^*, y^*)$ perturb either $x^*$ or $y^*$ but not both, by setting:

$$\alpha^{*} = \max(\alpha^{*}_c, \alpha^{*}_d) = (\max(\alpha^{*}_c, \alpha^{*}_d), \cdots, \max(\alpha^{*}_c, \alpha^{*}_d))^T \tag{46}$$

$$\beta^{*} = \max(\beta^{*}_c, \beta^{*}_d) = (\max(\beta^{*}_c, \beta^{*}_d), \cdots, \max(\beta^{*}_c, \beta^{*}_d))^T, \tag{47}$$

we conclude, based on (44) and (45), that the second inequality in (20) is satisfied for all $(x, y) \in \mathcal{N}_m(x^*, y^*)$ and for any $\alpha^{**} > \alpha^{*} \geq 0$ and $\beta^{**} > \beta^{*} \geq 0$.

“$\Leftarrow$” part: Assuming (20) is satisfied, we need to prove that $(x^*, y^*)$ is a $CLM_m$. The proof is straightforward and is similar to that of Theorem 1.

Artificial Intelligence (accepted to appear)
Appendix D: Proof of Theorem 5

We prove the theorem by showing the equivalence of (25) and the combined (26) and (27).

“⇒” part: Given $z^*$ that satisfies (25), we show that it also satisfies (26) and (27). Since for all $t = 0, \ldots, N$, any $z \in \mathcal{N}_p^{(t)}(z^*)$ is also a point in $\mathcal{N}_p(z^*)$; hence, the second inequality in (26) is implied by the second inequality in (25). The first inequality in (26) and the inequality in (27) are obvious, as all the constraints are satisfied at $z^*$.

“⇐” part: We prove this part by contradiction. Assuming that $z^*$ satisfies (26) and (27) but not (25), the first inequality in (25) cannot be violated because the first inequality in (26) and the inequality in (27) imply that all the local and global constraints are satisfied. Therefore, it must be the second inequality in (25) that is not satisfied at $z^*$. That is, there exist $z \in \mathcal{N}_p(z^*)$ and a unique $t'$ where $z \in \mathcal{N}_b^{(t')}(z^*)$ (according to the definition of $\mathcal{N}_p(z)$ in (23)) such that:

$$L_m(z^*, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \not\preceq L_m(z, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}).$$

This implies that the second inequality in (26) is not satisfied at $t = t'$, which contradicts our assumption that $z^*$ satisfies (26) and (27). Our argument proves that any $z^*$ that satisfies (26) and (27) must also satisfy (25).

Acknowledgments

The authors would like to thank the two reviewers who provided comprehensive and detailed comments that help improve this paper.

References


*Artificial Intelligence* (accepted to appear)


Artificial Intelligence (accepted to appear)
REFERENCES


*Artificial Intelligence* (accepted to appear)
REFERENCES


Artificial Intelligence (accepted to appear)