Available online at www.sciencedirect.com

SCIENCE DIRECT®

Artificial Intelligence 170 (2006) 187–231

Artificial Intelligence

www.elsevier.com/locate/artint

Constraint partitioning in penalty formulations for solving temporal planning problems [☆]

Benjamin W. Wah [a,*], Yixin Chen [b]

[a] *Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign, Urbana, IL 61801, USA*

[b] *Department of Computer Science, Washington University, St. Louis, MO 63130, USA*

Received 25 August 2003; accepted 7 July 2005

Available online 24 August 2005

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

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

[*] Corresponding author.
E-mail address: wah@uiuc.edu (B.W. Wah).
URL: http://manip.crhc.uiuc.edu.

0004-3702/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2005.07.001

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: 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.

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:

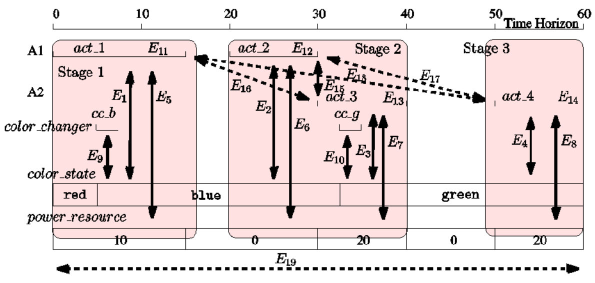
$$\begin{aligned} &(P_t)\colon \min_z \; J(z) \qquad (1)\\ &\text{subject to } h^{(t)}\big(z(t)\big)=0,\; g^{(t)}\big(z(t)\big)\leqslant 0,\; t=0,\ldots,N \text{ (local constraints)},\\ &\text{and } H(z)=0,\; G(z)\leqslant 0 \text{ (global constraints)}. \end{aligned}$$

Here, Stage t, $t=0,\ldots,N$, has local *state vector* $z(t)=(z_1(t),\ldots,z_{u_t}(t))^{\mathrm{T}}$ of u_t mixed variables; $h^{(t)}=(h_1^{(t)},\ldots,h_{m_t}^{(t)})^{\mathrm{T}}$ is a vector of m_t local equality-constraint functions that involve $z(t)$; $g^{(t)}=(g_1^{(t)},\ldots,g_{r_t}^{(t)})^{\mathrm{T}}$ is a vector of r_t local inequality-constraint functions of $z(t)$; $H=(H_1,\ldots,H_p)^{\mathrm{T}}$ is a vector of p global equality-constraint functions that involve $z=\bigcup_{i=0}^{N} z(i)$; and $G=(G_1,\ldots,G_q)^{\mathrm{T}}$ 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 Fig. 1 solved by ASPEN [9]. The problem involves scheduling four activities: *act*_1

```
model toy HORIZON_START = 1998-1/00:00:00; horizon_duration = 60s; time_scale = second;;
parameter string color 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_before 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;;
```



Local constraints (solid arrows):
- *color_state* constraints E_1-E_4 for *act_1*-*act_4*;
- *power_resource* constraints E_5-E_8 for *act_1*-*act_4*;
- *color_state* transition constraints E_9 and E_{10} relating *color_changer* and *color_state*;
- duration constraints E_{11}-E_{14} for *act_1*-*act_4*;
- *act_2* ends_before start of *act_3* by [0,30] (E_{15})

Global constraints (dashed arrows):
- *act_1* ends_before start of *act_3* by [0,30] (E_{16});
- *act_2* ends_before start of *act_4* by [0,30] (E_{17});
- *act_1* ends_before start of *act_4* by [0,30] (E_{18});
- *power_resource* always less than capacity of *power_resource* (E_{19}).

Fig. 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.

and *act*_2 of type A1 and *act*_3 and *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 Fig. 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 Fig. 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:

(c1) $w(t) \leqslant p(t) \leqslant 25$, $\forall t = 0, \ldots, 60$; // *power_resource* capacity constraint
(c2) $0 \leqslant s(act\_3) - e(act\_1) \leqslant 30$; // *act*_1 ends_before start of *act*_3 by $[0, 30]$
(c3) $s(act\_1) = t \Longrightarrow c(t) = 2$; $\forall t = 0, \ldots, 60$; // *color_state* constraint for *act*_1
(c4) $s(cc\_b) = t \Longrightarrow c(t) = 1$; $\forall t = 0, \ldots, 60$; // *color_changer cc_b* effect constraint

The constraints are either equality or inequality constraints (such as (c1) and (c2)), or deduction constraints (such as (c3) and (c4)). A deduction constraint $A \Rightarrow B$, where A and

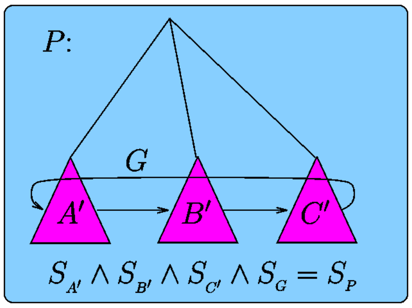


Fig. 2. An illustration of constraint partitioning that decomposes P into a conjunction ($\wedge$) of three subproblems and a set of global constraints to be resolved, where the complexity of each subproblem is substantially smaller than that of P. The set of global constraints G includes constraints in P that span across variables in multiple subproblems and new constraints added to maintain the consistency of shared entities and variables across subproblems.

B are equality or inequality constraints, can be encoded as an equivalent equality constraint $H(A \Rightarrow B) = 0$:

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

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

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, intelligent 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 assignment 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 ($\wedge$) of subproblems, each with local constraints and related to others by global constraints (Fig. 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 their consistency. This approach is attractive for solving temporal planning problems because many of their constraints and objectives are related to activities with temporal locality, and the constraints can be partitioned into independent subproblems related by only a small number of global constraints.

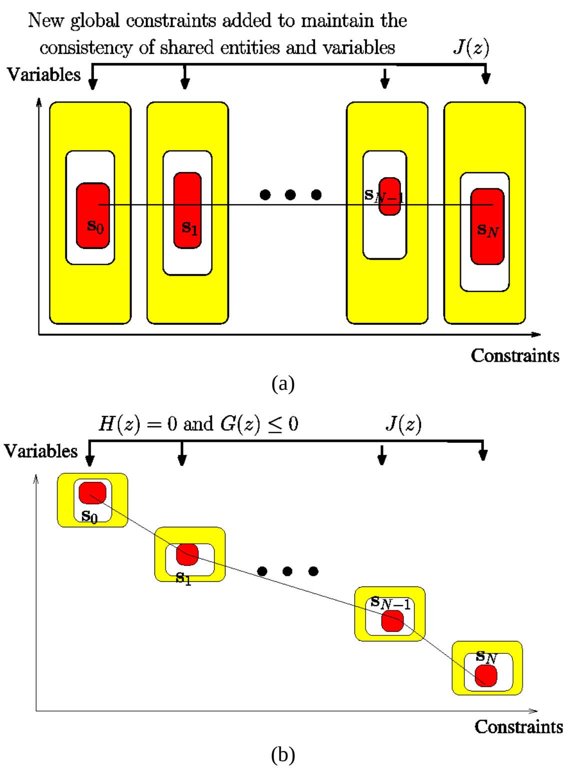

New global constraints added to maintain the consistency of shared entities and variables
$J(z)$
Variables
s_0
s_1
s_{N-1}
s_N
Constraints
(a)
$H(z) = 0$ and $G(z) \le 0$
$J(z)$
Variables
s_0
s_1
s_{N-1}
s_N
Constraints
(b)

Fig. 3. Two extreme configurations when partitioning the constraints of a problem. (a) Totally overlapped variable sets. (b) Totally disjoint variable sets.

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.

Fig. 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

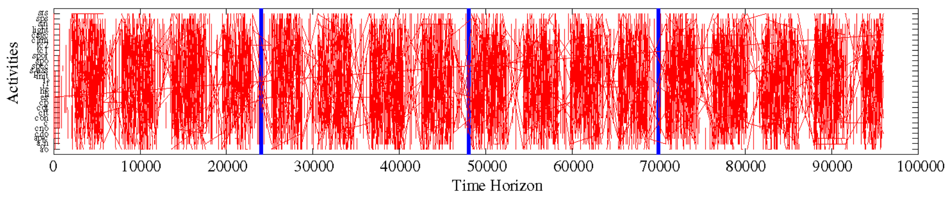


Fig. 4. The 3687 constraints of an initial infeasible schedule generated by ASPEN in solving CX1-PREF with 16 orbits. Each constraint is shown as a line that relates two activities (labeled in the y-axis) scheduled at two time instances in the horizon (x-axis). The partitioning of the constraints into four stages (separated by bold vertical lines) leads to 3580 local constraints and 107 global constraints. A local constraint remains associated with a stage even when activities are rescheduled. The number of iterations to find a better solution to the problem is reduced from 12,043 taken by ASPEN to 1102 after partitioning.

disjoint variable 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 $\mathbf{s}_i$ be the set of points that satisfy ESPC in stage i, $i = 0, \ldots, N$.

In Fig. 3(a), 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 $|\mathbf{s}_0 \cap \mathbf{s}_1 \cap \cdots \cap \mathbf{s}_N|$. Since the number of such global constraints as well as $\mathbf{s}_i$, $i = 1, \ldots, N$, can be large, the complexity for resolving the global constraints can be very high. In contrast, in Fig. 3(b), 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 $|\mathbf{s}_0| \cdot |\mathbf{s}_1| \cdots |\mathbf{s}_N|$. 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, Fig. 1 (*respectively* Fig. 4) shows the nineteen (*respectively* 3687) constraints of an initial infeasible schedule generated by ASPEN [9] in solving the toy example (*respectively*

CX1-PREF with sixteen orbits). After partitioning the constraints into three (*respectively* four) stages, the resulting problem has fifteen (*respectively* 3580) local constraints and four (*respectively* 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 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 Fig. 1 (*respectively* 4) can be solved by ASPEN in 16 (*respectively* 12,043) iterations and by our implementation in 12 (*respectively* 1102) iterations with the same (*respectively* 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

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

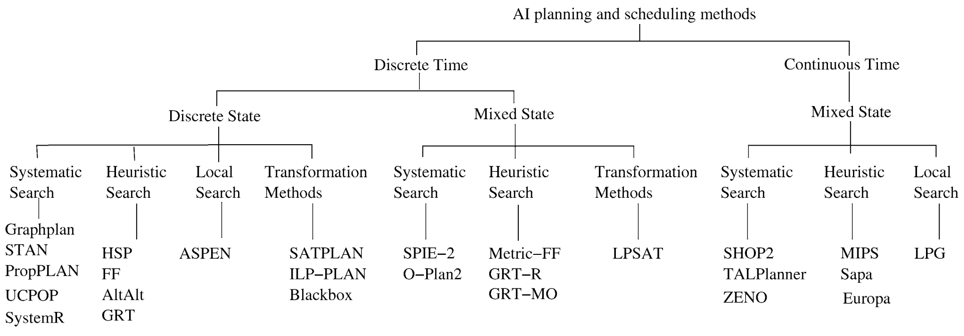


Fig. 5. A classification of existing planning and scheduling approaches.

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

$$(P_c)\colon \min_x \; f(x) \text{ where } x = (x_1, \ldots, x_v)^{\mathrm{T}} \in \mathbb{R}^v \tag{2}$$
$$\text{subject to } h(x) = \big(h_1(x), \ldots, h_m(x)\big)^{\mathrm{T}} = 0 \text{ and } g(x) = \big(g_1(x), \ldots, g_r(x)\big)^{\mathrm{T}} \leqslant 0,$$

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 $\mathcal{N}_c(x^*) = \{x'\colon \|x' - x^*\| \leqslant \varepsilon \text{ and } \varepsilon \to 0\}$, the *continuous neighborhood* of x^*, where $\varepsilon \to 0$ means that ε is arbitrarily close to 0.

Definition 1. Point x^* is a *CLM_c*, a constrained local minimum of P_c with respect to points in $\mathcal{N}_c(x^*)$, if x^* is feasible and $f(x^*) \leqslant f(x)$ 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 $\lambda = (\lambda_1, \ldots, \lambda_m)^{\mathrm{T}} \in \mathbb{R}^m$ and $\mu = (\mu_1, \ldots, \mu_r)^{\mathrm{T}} \in \mathbb{R}^r$, is defined as:

$$L(x, \lambda, \mu) = f(x) + \lambda^{\mathrm{T}} h(x) + \mu^{\mathrm{T}} g(x). \tag{3}$$

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,[1] then there exist unique $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^r$ such that:

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0 \tag{4}$$

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, λ and μ 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:

$$F(x, \lambda) = \begin{bmatrix} \nabla_x f(x) + \lambda^{\mathrm{T}} \nabla_x h(x) \\ h(x) \end{bmatrix} = 0, \tag{5}$$

[1] Point 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 $\nabla_x g_{a_1}(x), \ldots, \nabla_x g_{a_l}(x), a_i \in A(x)$ (the set of active inequality constraints) are linearly independent. An inequality constraint $g_i(x) \leqslant 0$ is active when $g_i(x) = 0$. It will affect the search direction only when it is active and can be ignored otherwise.

where $\nabla_x h(x)^{\mathrm{T}} = [\nabla_x h_1(x), \ldots, \nabla_x h_m(x)]$ 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 λ, μ 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, λ, and μ 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.[2]

(b) Sufficient *saddle-point condition* [2,26]. 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^*:

$$L(x^*, \lambda, \mu) \leqslant L(x^*, \lambda^*, \mu^*) \leqslant L(x, \lambda^*, \mu^*) \tag{6}$$

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 λ^* and μ^* that satisfy (6) at each CLM_c x^* of P_c.

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

$$\min_x \ f(x) = -x^2 \quad \text{subject to} \quad h(x) = x - 5 = 0. \tag{7}$$

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 $\nabla_x^2 L(x, \lambda)|_{x^*, \lambda^*} = -2 < 0$, we know that $L(x, \lambda)$ is at a local maximum with respect to x at (x^*, λ^*) instead of a local minimum. Hence, there exists no λ^* 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^*, λ^*, and μ^* 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 ℓ_1-penalty method based on the following ℓ_1-penalty function [17]:

$$\ell_1(z, c) = f(z) + c \cdot \max\bigl(0, |h_1(z)|, \ldots, |h_m(z)|, g_1(z), \ldots, g_q(z)\bigr). \tag{8}$$

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)

[2] Constraint 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].

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.

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:

$$L_p(x, \alpha, \beta) = f(x) + \alpha^{\mathrm{T}} P\bigl(h(x)\bigr) + \beta^{\mathrm{T}} Q\bigl(g(x)\bigr), \tag{9}$$

where P and Q are possible transformation functions. The goal of a penalty method is to find suitable α^* and β^* 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 α^* and β^* (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 α 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 α.

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 non-negative functions that satisfy the following properties: (a) $P(h(x)) \geqslant 0$ and $Q(g(x)) \geqslant 0$; and (b) $P(h(x)) = 0$ if and only if $h(x) = 0$, and $Q(g(x)) = 0$ if and only if $g(x) \leqslant 0$.

For any finite penalty vectors α^{**} and β^{**} larger than some thresholds, $\alpha^{**} > \alpha^*$[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 α and β 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 $-100 \leqslant x \leqslant 100$, 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 α^* depends on the range of x. For example, Fig. 6 show that, if $-1000 \leqslant x \leqslant 1000$, then there will be a global minimum of $L_p(x,\alpha^{**})$ at $x^* = 5$ for any $\alpha^{**} > \alpha^* = 1005$. This example shows that α^* 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.

[3] $\alpha^{**} > \alpha^*$ means that every element of α^{**} is larger than the corresponding element of α^*. Further, $\alpha^{**} \geqslant \alpha^*$ means that each element of α^{**} is larger than or equal to the corresponding element of α^*.

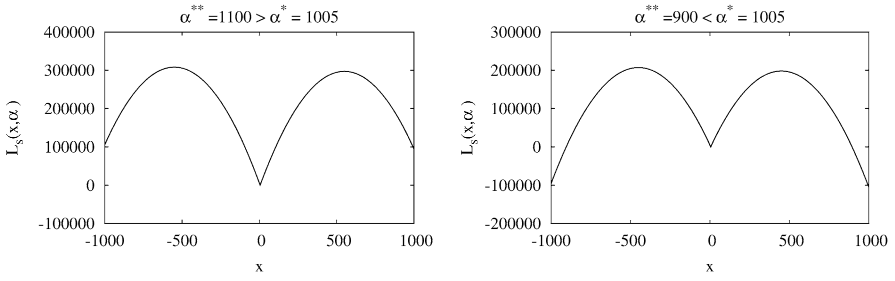


Fig. 6. An illustration of a global minimum of $L_p(x,\alpha^{**})$ at $x^* = 5$ for any $\alpha^{**} > \alpha^* = 1005$ but not one when $\alpha^{**} \leqslant \alpha^*$ for the CNLP problem in (7), where $-1000 \leqslant x \leqslant 1000$.

Instead of finding α^{**} and β^{**} 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 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 α^{**} and β^{**} 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 [24,42–44]. 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.

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:

$$L_c(x,\alpha,\beta) = f(x) + \alpha^{\mathrm T}\bigl|h(x)\bigr| + \beta^{\mathrm T}\max\bigl(0, g(x)\bigr), \tag{10}$$

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

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

$$D_x\bigl(\phi(x'), \vec{p}\bigr) = \lim_{\varepsilon \to 0} \frac{\phi(x' + \varepsilon\vec{p}) - \phi(x')}{\varepsilon}. \tag{11}$$

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

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

$$\begin{aligned} &\nexists \vec{p} \in X \quad \text{such that} \quad D_x\bigl(h_i(x^*), \vec{p}\bigr) = 0 \quad \text{and} \\ &D_x\bigl(g_j(x^*), \vec{p}\bigr) = 0 \quad \forall i \in C_h \text{ and } j \in C_g, \end{aligned} \tag{12}$$

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 $\vec{p}$. It rules out the case in which there is a direction $\vec{p}$ at x^* along which all equality constraints and active inequality constraints have zero subdifferentials. Intuitively, constraint qualification at x^* ensures the existence of finite α and β that lead to a local minimum of (10) at x^*. Consider a neighboring point $x^* + \vec{p}$ infinitely close to x^*, where the objective function f at x^* decreases along $\vec{p}$ and all active constraints at x^* have zero subdifferentials along $\vec{p}$. In this case, since all the active constraints at $x^* + \vec{p}$ are also satisfied, it will be impossible to find finite α and β in order to establish a local minimum of (10) at x^* with respect to $x^* + \vec{p}$. To ensure a local minimum of (10) at x^*, the above scenario must not be true for any $\vec{p}$ at x^*.

Note that our condition is less restricted than the regularity condition in KKT, which requires the linear independence of the gradients of the equality and active inequality constraint functions.

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

Theorem 1 (Necessary and sufficient ESPC on *CLM*$_c$ of P_c). *Suppose $x^* \in \mathbb{R}^v$ 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 $\alpha^* \geqslant 0$ and $\beta^* \geqslant 0$ such that the following is satisfied*:

$$L_c(x^*, \alpha, \beta) \leqslant L_c(x^*, \alpha^{**}, \beta^{**}) \leqslant L_c(x, \alpha^{**}, \beta^{**}) \tag{13}$$

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 α and β. One approach to look for an extended saddle point of L_c is to increase gradually α^{**} and β^{**}, 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 α^* and β^* that satisfy Theorem 1 can be much smaller than the corresponding α^* and β^* found by the dynamic-penalty method in Section 2.2.

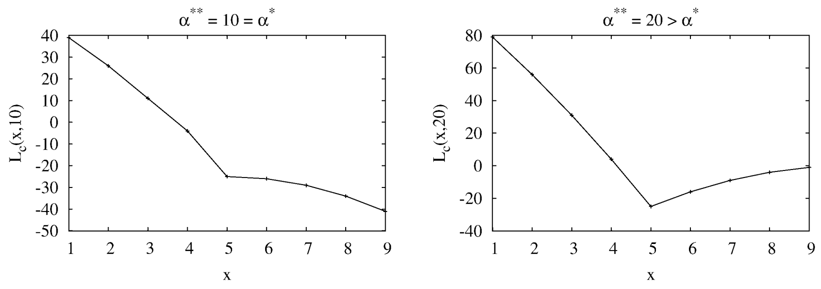


Fig. 7. An illustration that (13) is satisfied when $\alpha^{**} > \alpha^* = 10$ for the CNLP problem in (7). $L_c(x, \alpha^{**})$ is a strict local minimum around $x^* = 5$ when $\alpha^{**} > \alpha^*$ but is not one when $\alpha^{**} = \alpha^*$.

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 $-1000 \leqslant x \leqslant 1000$ in the dynamic-penalty method (Fig. 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. Fig. 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 α^{**} 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.

$$(P_d): \min_y \ f(y) \text{ where } y = (y_1, \ldots, y_w)^{\mathrm{T}} \in \mathbb{D}^w \tag{14}$$
$$\text{subject to } h(y) = 0 \text{ and } g(y) \leqslant 0.$$

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 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 $y' \in \mathcal{N}_d(y) \Leftrightarrow y \in \mathcal{N}_d(y')$, 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 $f(y^*) \leqslant f(y)$ 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:

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

Theorem 2 (Necessary and sufficient ESPC on CLM_d of P_d [51,55]). *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 $\alpha^* \geqslant 0$ and $\beta^* \geqslant 0$ such that the following condition is satisfied*:

$$L_d(y^*,\alpha,\beta) \leqslant L_d(y^*,\alpha^{**},\beta^{**}) \leqslant L_d(y,\alpha^{**},\beta^{**}) \tag{16}$$

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:

$$(P_m)\colon \min_{x,y} f(x,y) \text{ where } x = (x_1,\ldots,x_v)^{\mathrm{T}} \in \mathbb{R}^v \text{ and } y = (y_1,\ldots,x_w)^{\mathrm{T}} \in \mathbb{D}^w \quad \text{subject to } h(x,y) = 0 \text{ and } g(x,y) \leqslant 0. \tag{17}$$

The goal of solving P_m is to find a constrained local minimum (x^*, y^*) with respect to $\mathcal{N}_m(x^*, y^*)$, 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.

Definition 8. $\mathcal{N}_m(x,y)$, the mixed neighborhood of $(x,y) \in \mathbb{R}^v \times \mathbb{D}^w$ in mixed space, is made up of the union of the continuous neighborhood and the user-defined discrete neighborhood:

$$\begin{aligned}\mathcal{N}_m(x,y) &= \mathcal{N}_c(x)|_y \cup \mathcal{N}_d(y)|_x \\ &= \big\{(x',y) \mid x' \in \mathcal{N}_c(x)\big\} \cup \big\{(x,y') \mid y' \in \mathcal{N}_d(y)\big\}.\end{aligned} \tag{18}$$

Definition 9. Point (x^*, y^*) is a *CLM*$_m$, a constrained local minimum of P_m with respect to points in $\mathcal{N}_m(x^*, y^*)$, if (x^*, y^*) is feasible and $f(x^*, y^*) \leqslant f(x, y)$ for all feasible $(x, y) \in \mathcal{N}_m(x^*, y^*)$.

Definition 10. The penalty function of P_m is defined as in (9) by transforming the constraint functions of P_m into non-negative functions:

$$L_m(x, y, \alpha, \beta) = f(x, y) + \alpha^{\mathrm{T}}\big|h(x, y)\big| + \beta^{\mathrm{T}} \max\big(0, g(x, y)\big) \quad \text{where } \alpha \in \mathbb{R}^m \text{ and } \beta \in \mathbb{R}^r. \tag{19}$$

Theorem 3 (Necessary and sufficient ESPC on *CLM*$_m$ of P_m). *Suppose $(x^*, y^*) \in \mathbb{R}^v \times \mathbb{D}^w$ is a point in the mixed search space of P_m, and x^* satisfies the constraint qualification condition in* (12) *for given y^*, then (x^*, y^*) is a CLM$_m$ of P_m if and only if there exist finite $\alpha^* \geqslant 0$ and $\beta^* \geqslant 0$ such that the following condition is satisfied*:

$$L_m(x^*, y^*, \alpha, \beta) \leqslant L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leqslant L_m(x, y, \alpha^{**}, \beta^{**}) \tag{20}$$

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

The proof of the theorem is shown in Appendix C.

The following corollary facilitates the search of points that satisfy (20) by decomposing the condition into two independent necessary conditions. It follows directly from (18), which defines $\mathcal{N}_m(x, y)$ to be the union of points perturbed in either the discrete or the continuous subspace. Such decomposition cannot be accomplished if a mixed neighborhood like $\mathcal{N}_c(x) \times \mathcal{N}_d(y)$ were used.

Corollary 1. *Given the definition of $\mathcal{N}_m(x, y)$ in* (18), *the ESPC in* (20) *can be rewritten into two necessary conditions that, collectively, are sufficient*:

$$L_m(x^*, y^*, \alpha, \beta) \leqslant L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leqslant L_m(x^*, y, \alpha^{**}, \beta^{**}) \quad \textit{where } y \in \mathcal{N}_d(y^*)|_{x^*}, \tag{21}$$

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leqslant L_m(x, y^*, \alpha^{**}, \beta^{**}) \quad \textit{where } x \in \mathcal{N}_c(x^*)|_{y^*}. \tag{22}$$

In summary, we have presented in this section a set of necessary and sufficient conditions that govern all constrained local minima in nonlinear continuous, discrete, and mixed optimization problems. In contrast to general penalty approaches, α^{**} and β^{**} always exist in ESPC for any constrained local minimum, provided that the constraint qualification condition is satisfied in the continuous subspace. The similarity of these three conditions allows problems in these three classes to be solved in a unified fashion.

3.2. Search procedures for finding extended saddle points

As is discussed in the last section, a *CLM*$_c$ of P_c can be found by gradually increasing α^{**} and β^{**}, 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 Fig. 8(a). (The algorithm for solving

```
procedure ESP_search_continuous($P_c$, $x$, $\alpha^{\max}$, $\beta^{\max}$);
  $\alpha \leftarrow 0$; $\beta \leftarrow 0$;
  repeat
    for ($i = 1, \ldots, m$) if ($h_i(x) \neq 0$ and $\alpha_i < \alpha_i^{\max}$) then increase $\alpha_i$ by $\delta$;
    for ($j = 1, \ldots, r$) if ($g_j(x) \nleq 0$ and $\beta_j < \beta_j^{\max}$) then increase $\beta_j$ by $\delta$;
    repeat
      perform descent of $L_c(x, \alpha, \beta)$ with respect to $x$;
    until a local minimum of $L_c(x, \alpha, \beta)$ is found;
  until ($\alpha_i > \alpha_i^{\max}$ for all $h_i(x) \neq 0$ and $\beta_j > \beta_j^{\max}$ for all $g_j(x) \nleq 0$)
    or a $CLM_c$ of $P_c$ is found;
  return $CLM_c$ if found;
end_procedure
```

(a)

```
procedure ESP_search_mixed($P_m$, $z$, $\alpha^{\max}$, $\beta^{\max}$);
  $\alpha \leftarrow 0$; $\beta \leftarrow 0$;
  repeat
    for ($i = 1, \ldots, m$) if ($h_i(z) \neq 0$ and $\alpha_i < \alpha_i^{\max}$) then increase $\alpha_i$ by $\delta$;
    for ($j = 1, \ldots, r$) if ($g_j(z) \nleq 0$ and $\beta_j < \beta_j^{\max}$) then increase $\beta_j$ by $\delta$;
    repeat
      perform descent of $L_m(z, \alpha, \beta)$ with respect to $x$ for given $y$;
    until a local minimum of $L_m(z, \alpha, \beta)$ with respect to $x$ for given $y$ is found;
    repeat
      perform descent of $L_m(z, \alpha, \beta)$ with respect to $y$ for given $x$;
    until a local minimum of $L_m(z, \alpha, \beta)$ with respect to $y$ for given $x$ is found;
  until ($\alpha_i > \alpha_i^{\max}$ for all $h_i(z) \neq 0$ and $\beta_j > \beta_j^{\max}$ for all $g_j(z) \nleq 0$)
    or a $CLM_m$ of $P_m$ is found;
  return $CLM_m$ if found;
end_procedure
```

(b)

Fig. 8. Iterative procedures to look for CLM_c of P_c and CLM_m of P_m. The bounds on α and β, $\alpha^{\max}$ and $\beta^{\max}$, are user-provided. (a) Direct implementation of (13) to look for CLM_c of P_c for given starting point x. (b) Direct implementation of (21) and (22) to look for CLM_m of P_m for given starting point $z = (x, y)$.

P_d is similar and is not shown.) Assuming α^{**} and β^{**} 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 solution to P_c is not found at the local minimum x of $L_c(x, \alpha^{**}, \beta^{**})$, the penalties corresponding to the violated constraints are increased. The process is repeated until a CLM_c is found or when α^{**} (*respectively* β^{**}) is larger than the user-provided maximum bound $\alpha^{\max}$ (*respectively* $\beta^{\max}$), where $\alpha^{\max}$ (*respectively* $\beta^{\max}$) is chosen to be so large that it exceeds α^* (*respectively* β^*).

Fig. 8(b) shows the pseudo code which solves P_m by looking for x^*, y^*, α^{**}, and β^{**} that satisfy Corollary 1. By performing descents of $L_m(x, y, \alpha, \beta)$ in the continuous and discrete neighborhoods in the two inner loops, it looks for a local minimum (x^*, y^*) of $L_m(x, y, \alpha, \beta)$ with respect to points in $\mathcal{N}_m(x, y)$. The outer loop increases the penalties of violated constraints and stops when a CLM_m is found or when α^{**} (*respectively* β^{**}) exceeds its maximum bound $\alpha^{\max}$ (*respectively* $\beta^{\max}$).

Because $L_c(x, \alpha^{**}, \beta^{**})$ and $L_m(x, y, \alpha^{**}, \beta^{**})$ may have many local minima and some of them do not correspond to constrained local minima even when $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$, it is possible for the iterative procedures in Fig. 8 to terminate without finding a constrained local minimum. The following theorem summarizes this observation.

Theorem 4. *When $\alpha^{\max} > \alpha^*$ and $\beta^{\max} > \beta^*$, the iterative procedure in Fig.* 8(*a*) (respectively 8(*b*)) *generates fixed points that are necessary but not sufficient to satisfy* (13) (respectively (21) *and* (22)).

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

First, when α^{**} and β^{**} 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 α^{**} and β^{**} 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 α^{**} and β^{**} 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 α or β 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:

Definition 11. $\mathcal{N}_p(z)$, the *mixed neighborhood of* z for partitioned problem P_t, is defined as:

$$\mathcal{N}_p(z) = \bigcup_{t=0}^{N} \mathcal{N}_p^{(t)}(z) = \bigcup_{t=0}^{N} \bigl\{ z' \mid z'(t) \in \mathcal{N}_m\bigl(z(t)\bigr),\ \text{and } z'_i(s) = z_i(s) \\ \forall z_i(s) \notin z(t),\ s \neq t,\ i = 1, \dots, u_s \bigr\},$$

where $\mathcal{N}_m(z(t))$ is the mixed-space neighborhood of $z(t)$ in Stage t.

Intuitively, $\mathcal{N}_p(z)$ 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 $\mathcal{N}_p(z)$ 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 $\Phi(z,\gamma,\eta) = \gamma^{\mathrm{T}}|H(z)| + \eta^{\mathrm{T}} \max(0, G(z))$ be the sum of the transformed global constraint functions weighted by their penalties, where $\gamma = (\gamma_1, \dots, \gamma_p)^{\mathrm{T}} \in \mathbb{R}^p$ and $\eta = (\eta_1, \dots, \eta_q)^{\mathrm{T}} \in \mathbb{R}^q$ 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:

$$L_m(z,\alpha,\beta,\gamma,\eta) = J(z) + \sum_{t=0}^{N} \Bigl\{ \alpha(t)^{\mathrm{T}} \bigl| h^{(t)}\bigl(z(t)\bigr)\bigr| + \beta(t)^{\mathrm{T}} \max\bigl(0, g^{(t)}\bigl(z(t)\bigr)\bigr) \Bigr\} \\ + \Phi(z,\gamma,\eta), \tag{23}$$

$$\Gamma_m\bigl(z,\alpha(t),\beta(t),\gamma,\eta\bigr) = J(z) + \alpha(t)^{\mathrm{T}} \bigl| h^{(t)}\bigl(z(t)\bigr)\bigr| \\ + \beta(t)^{\mathrm{T}} \max\bigl(0, g^{(t)}\bigl(z(t)\bigr)\bigr) + \Phi(z,\gamma,\eta), \tag{24}$$

where $\alpha(t) = (\alpha_1(t), \ldots, \alpha_{m_t}(t))^{\mathrm{T}} \in \mathbb{R}^{m_t}$ and $\beta(t) = (\beta_1(t), \ldots, \beta_{r_t}(t))^{\mathrm{T}} \in \mathbb{R}^{r_t}$ 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 $\mathcal{N}_p(z)$ if and only if there exist finite $\alpha^* \geqslant 0$, $\beta^* \geqslant 0$, $\gamma^* \geqslant 0$, and $\eta^* \geqslant 0$ such that the following ESPC is satisfied*:

$$L_m(z^*, \alpha, \beta, \gamma, \eta) \leqslant L_m(z^*, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \leqslant L_m(z, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \tag{25}$$

for any $\alpha^{**} > \alpha^*$, $\beta^{**} > \beta^*$, $\gamma^{**} > \gamma^*$ and $\eta^{**} > \eta^*$, and for all $\alpha \in \mathbb{R}^{\sum_{i=0}^{N} m_i}$, $\beta \in \mathbb{R}^{\sum_{i=0}^{N} r_i}$, $\gamma \in \mathbb{R}^p$, $\eta \in \mathbb{R}^q$, and $z \in \mathcal{N}_p(z^*)$.

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 $\mathcal{N}_p(z)$, the ESPC in* (25) *can be rewritten into $N+2$ necessary conditions that, collectively, are sufficient*:

$$\begin{aligned} \Gamma_m\big(z^*, \alpha(t), \beta(t), \gamma^{**}, \eta^{**}\big) &\leqslant \Gamma_m\big(z^*, \alpha(t)^{**}, \beta(t)^{**}, \gamma^{**}, \eta^{**}\big) \\ &\leqslant \Gamma_m\big(z, \alpha(t)^{**}, \beta(t)^{**}, \gamma^{**}, \eta^{**}\big), \end{aligned} \tag{26}$$

$$L_m(z^*, \alpha^{**}, \beta^{**}, \gamma, \eta) \leqslant L_m(z^*, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \tag{27}$$

for all $z \in \mathcal{N}_p^{(t)}(z^*)$, $\alpha(t) \in \mathbb{R}^{m_t}$, $\beta(t) \in \mathbb{R}^{r_t}$, and $t = 0, \ldots, N$.

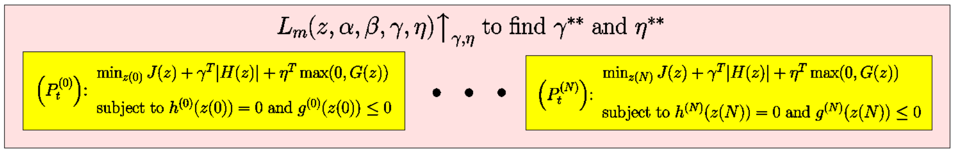
The proof is shown in Appendix D.

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 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:

$$\begin{aligned} &(P_t^{(t)}):\ \min_{z(t)}\ J^{(t)}(z) = J(z) + \gamma^{\mathrm{T}}\big|H(z)\big| + \eta^{\mathrm{T}} \max\big(0, G(z)\big) \\ &\text{subject to } h^{(t)}\big(z(t)\big) = 0 \text{ and } g^{(t)}\big(z(t)\big) \leqslant 0. \end{aligned} \tag{28}$$

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).


$L_m(z,\alpha,\beta,\gamma,\eta)\uparrow_{\gamma,\eta}$ to find γ^{**} and η^{**}
$\left(P_t^{(0)}\right)$: $\min_{z(0)} J(z) + \gamma^T|H(z)| + \eta^T \max(0, G(z))$
subject to $h^{(0)}(z(0)) = 0$ and $g^{(0)}(z(0)) \le 0$
• • •
$\left(P_t^{(N)}\right)$: $\min_{z(N)} J(z) + \gamma^T|H(z)| + \eta^T \max(0, G(z))$
subject to $h^{(N)}(z(N)) = 0$ and $g^{(N)}(z(N)) \le 0$

(a)

```
**procedure** ESP_partitioned_search_mixed($P_t$, $z$, $\alpha^{\max}$, $\beta^{\max}$, $\gamma^{\max}$, $\eta^{\max}$);
  $\gamma \leftarrow 0$; $\eta \leftarrow 0$;
  **repeat**      // increase the penalties on violated global constraints //
    **for** ($i = 1, \ldots, p$) **if** ($H_i(z) \neq 0$ **and** $\gamma_i < \gamma_i^{\max}$) **then** increase $\gamma_i$ by $\delta$;
    **for** ($j = 1, \ldots, q$) **if** ($G_j(z) \nleq 0$ **and** $\eta_j < \eta_j^{\max}$) **then** increase $\eta_j$ by $\delta$;
    **for** $t = 0$ **to** $N$ // iterate over all $N+1$ stages to solve $P_t^{(t)}$ in each stage //
      apply an existing solver or call ESP_search_ mixed($P_t^{(t)}$, $z$, $\alpha^{\max}$, $\beta^{\max}$) to solve $P_t^{(t)}$
    **end_for**;
  **until** ($\gamma_i > \gamma_i^{\max}$ for all $H_i(z) \neq 0$ **and** $\eta_j > \eta_j^{\max}$ for all $G_j(z) \nleq 0$) **or** a $CLM_m$ of $P_t$ is found.
  **return** $CLM_m$ if found;
**end_procedure**
```

(b)

Fig. 9. The partition-and-resolve procedure to look for CLM_m of P_t. The bounds $\alpha^{\max}$, $\beta^{\max}$, $\gamma^{\max}$, and $\eta^{\max}$ are user-provided. (a) Partitioned search to look for points that satisfy (26) and (27). (b) Implementation for finding CLM_m of P_t that satisfies (26) and (27) for given starting point z.

4.2. The partition-and-resolve procedure

Fig. 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 Fig. 9(b) solves $P_t^{(t)}$ by looking for an extended saddle point that satisfies (26). This can be done by the procedure in Fig. 8(b), using fixed γ and η 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 γ and η exceed their maximum bounds. Similar to the result in Theorem 4, the procedure in Fig. 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 SGPlant$_t$(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 shuttle payload that is managed by the University of Colorado at Boulder. (c) OPTIMIZE and PREF are two benchmarks developed at JPL that come with the licensed release of ASPEN.

Implementation of the partition-and-resolve search. Based on Fig. 9, we have implemented SGPlan$_t$(ASPEN, N, repartitioning_strategy) [8]. In our implementation, we set

```
 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.       num_descents $\leftarrow 1$;
 6.       for $t = 1$ to $N$
 7.          for $k = 1$ to num_descents
 8.             call ASPEN to solve $P_t^{(t)}$ 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.       num_descents $\leftarrow$ min(100, 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
```

Fig. 10. SGPlan$_t$(ASPEN, N, repartitioning_strategy): The partition-and-resolve procedure used in SGPlan 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 Γ_m.

the weight of $J(z)$ in $P_t^{(t)}$ 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 Γ_m (line 8 of Fig. 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 Γ_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 1000 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$_t$(ASPEN, N, STATIC$_P$) partitions the horizon statically and evenly into N

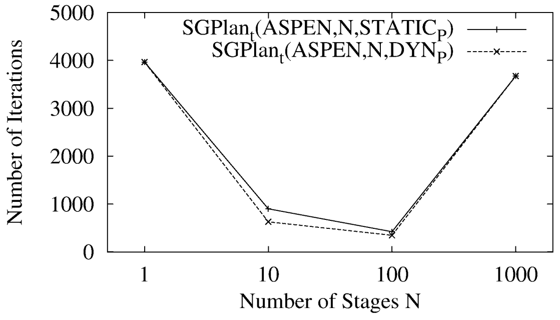


Fig. 11. Number of iterations taken by static and dynamic partitioning to find a feasible plan in the 8-orbit CX1-PREF problem.

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$_t$(ASPEN, N, DYN$_P$) 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, Fig. 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. Fig. 12 compares the performance of ASPEN, SGPlant$_t$(ASPEN, 100, STATIC$_P$), SGPlan$_t$(ASPEN, 1, STATIC$_P$) (a version of our planner without partitioning), and SGPlan$_t$(ASPEN, 100, DYN$_P$) 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$_t$ 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$_t$(ASPEN, 1, STATIC$_P$) 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$_t$(ASPEN, 100, STATIC$_P$) is able to find schedules of the same quality one to two orders of magnitude faster than ASPEN and SGPlan$_t$(ASPEN, 1, STATIC$_P$), as well as much better schedules when they converge.

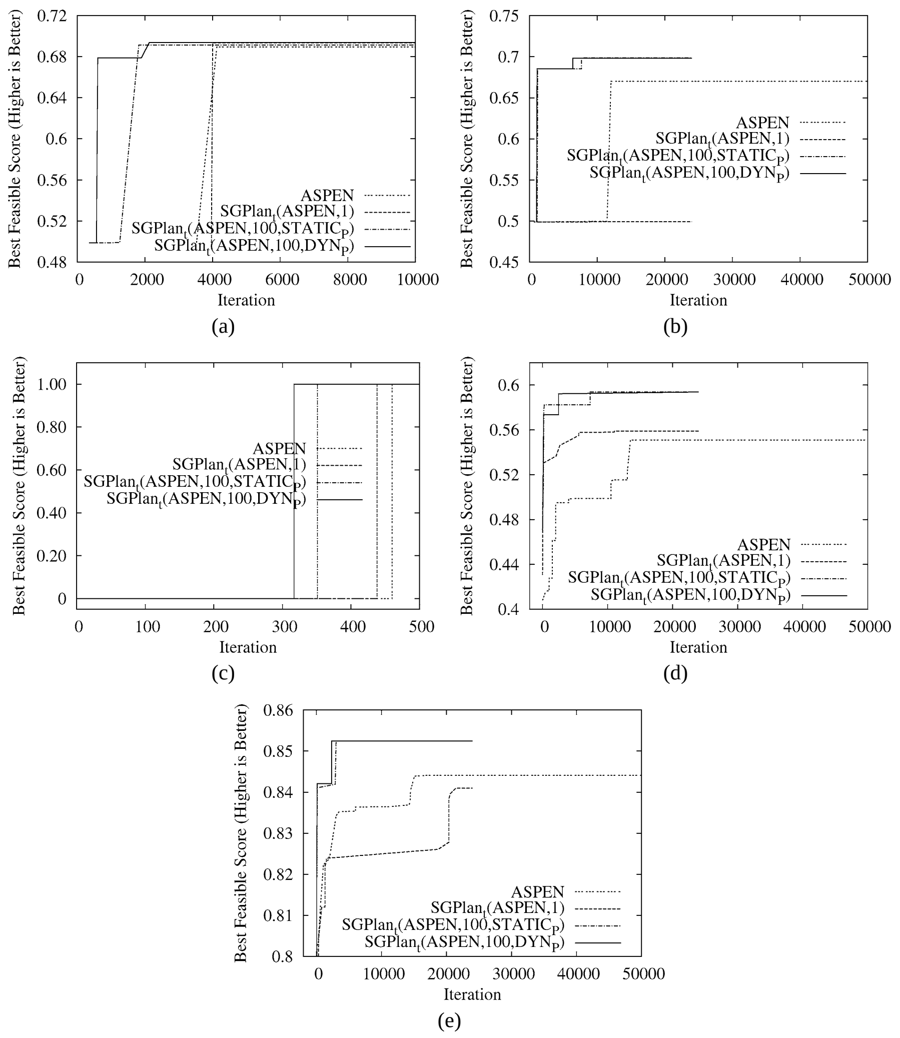


Fig. 12. Quality-time comparisons of $\mathrm{SGPlan_t}(\mathrm{ASPEN}, 1, \mathrm{STATIC_P})$, $\mathrm{SGPlan_t}(\mathrm{ASPEN}, 100, \mathrm{STATIC_P})$, ASPEN, and $\mathrm{SGPlan_t}(\mathrm{ASPEN}, 100, \mathrm{DYN_P})$. (All runs involving $\mathrm{SGPlan_t}$ were terminated at 24,000 iterations.) (a) CX1-PREF with 8 orbits. (b) CX1-PREF with 16 orbits. (c) DCAPS. (d) PREF. (e) OPTIMIZE.

Further, dynamic partitioning can lead to better schedules in shorter times than those of static partitioning. Hence, we conclude that improvements in $\mathrm{SGPlan_t}$ are mainly due to partitioning and not to annealing.

5.2. $\mathrm{SGPlan_t}(\mathrm{MIPS})$*: 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_I to goal state $s_G \in \mathcal{G}$ 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. Fig. 13 shows the pseudo code of $\text{SGPlan}_\text{t}(\text{MIPS}, N)$. 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:

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

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

$$\text{Group } 3 = (\textit{at plane}_1\ \textit{city}_0,\ \textit{at plane}_1\ \textit{city}_1,\ \textit{at plane}_1\ \textit{city}_2).$$

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

Based on the definition of symmetry groups, we define the *neighborhood* $\mathcal{N}_m(s)$ of s to include s and all states s' that differ from s by exactly one fact, where the facts that differ

```
1.  procedure SGPlan_t(MIPS, $N$)
2.    generate initial plan using relaxed operators;
3.    repeat
4.      iter ← 0;
5.      for $t = 0$ to $N$ // initial state $s_i(t)$ in Stage $t$ from initial plan //
6.        num_trials ← 0;
7.        repeat
8.          num_trials ← num_trials + 1;
9.          generate a new initial state in $\mathcal{N}_m(s_i(t))$ for Stage $t$;
10.         call MIPS to solve $P_t^{(t)}$ in Stage $t$;
11.         evaluate $J^{(t)}(z)$ in (28) of the solution plan generated by MIPS;
12.       until $J^{(t)}(z)$ is improved or num_trials > max_trials;
13.     end_for
14.     update penalty vector $\gamma$ on violated global constraints;
15.     iter ← iter + 1;
16.     if (iter mod $\tau$ is 0) then repartition the stages end_if;
17.   until no change in $z$ and $\gamma$ in an iteration;
18.   return the best plan found;
19. end_procedure
```

Fig. 13. SGPlan$_\mathrm{t}$(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.

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 difference between their numerical parts if $s \neq q$:

$$D(s,q) = (\text{number of different facts between } s \text{ and } q) + \sum_{i=1}^{n_r} \frac{|s_{r_i} - q_{r_i}|}{\max(s_{r_i}, q_{r_i})}\bigg|_{s\neq q}. \tag{29}$$

Hence, $D(s,q) = 0$ if and only if s and q are identical states. We further define $\mathcal{S}(s)$, 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 $q \in \mathcal{S}(s)$. Last, we define the *transition distance* $T(s,q)$ to be the minimum distance between s and q over all successors of s:

$$T(s,q) = \min_{v \in \mathcal{S}(s)} D(v,q). \tag{30}$$

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 $P_t^{(t)}$ in Stage t. After partitioning, $P_t^{(t)}$ has initial state $s_i(t)$ and goal state $s_i(t+1)$. (See Fig. 14 for the states defined in Stage t.) Since this initial local plan may be infeasible, we need to formulate $P_t^{(t)}$ that, when solved, will

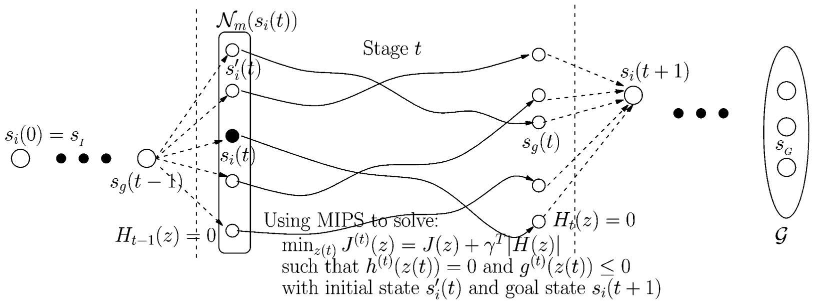

$\mathcal{N}_m(s_i(t))$
Stage t
$s_i'(t)$
$s_i(t)$
$s_i(0) = s_I$
$s_g(t-1)$
$H_{t-1}(z) = 0$
$s_i(t+1)$
$s_g(t)$
$H_t(z) = 0$
s_G
$\mathcal{G}$
Using MIPS to solve:
$\min_{z(t)} J^{(t)}(z) = J(z) + \gamma^T|H(z)|$
such that $h^{(t)}(z(t)) = 0$ and $g^{(t)}(z(t)) \le 0$
with initial state $s_i'(t)$ and goal state $s_i(t+1)$

Fig. 14. Formulation of the planning subproblem $P_t^{(t)}$ in Stage t to be solved by MIPS.

hopefully make the overall planning problem feasible (line 10 of Fig. 13). This subproblem has the same domain specification as the original problem, initial state $s_i'(t) \in \mathcal{N}_m(s_i(t))$, and goal state $s_i(t+1)$. In addition, there are two global constraints at the boundaries between Stage t and the predecessor and successor stages:

$$H_{t-1}(z) = T\big(s_g(t-1), s_i'(t)\big) = 0; \qquad H_t(z) = T\big(s_g(t), s_i(t+1)\big) = 0. \tag{31}$$

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

$$J^{(t)}(z) = J(z) + \gamma_{t-1} H_{t-1}(z) + \gamma_t H_t(z), \tag{32}$$

where γ_{t-1} and γ_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 Fig. 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:

$$\gamma_t \leftarrow \gamma_t + \omega \cdot H_t(z), \quad t = 0, 1, \ldots, N. \tag{33}$$

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 (τ in Fig. 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 $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ 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 SGPlan_t.[4]

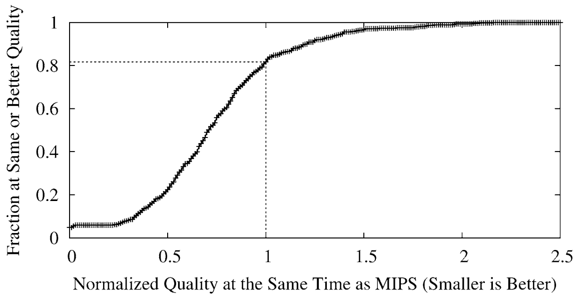
We conducted our experiments on an AMD Athlon MP2000 PC with Redhat Linux 7.2. In our experiments on MIPS and LPG, we used the August-2003 version of their executables with default parameters downloaded from their Web sites. In accordance to the way that planners were run in the International Planning Competitions, we used a fixed random seed of 1000 in LPG, MIPS, and $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$, making them behave like deterministic planners. We also used the same parameters specified for the original MIPS in the version of MIPS embedded in $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$. We then ran each planner once on each problem instance for a maximum time limit of 1000 sec.

For the 120 (out of a total of 160) instances solvable by MIPS, Fig. 15(a) compares the quality of the solution of each instance found by MIPS and by $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ when it was given the same amount of time taken by MIPS to solve that instance. It measures the fraction of instances that $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ found a better solution using the same amount of time taken by MIPS. In contrast, Fig. 15(b) compares the time taken by MIPS to solve an instance and that by $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ when it found a solution of the same or better quality as MIPS for that instance. It measures the fraction of instances that $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ 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 $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ could solve but MIPS could not find any feasible plan in 1000 sec. The results show that $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ 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 $\text{SGPlan}_\text{g}(\text{MIPS})$ 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].

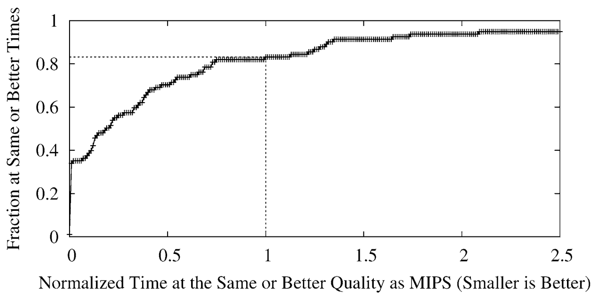
Of the 150 of the 160 instances solvable by $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$, $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ can find a feasible solution with better time (*respectively* quality) than MIPS in 94.4% (*respectively* 93.8%).

As a reference, Fig. 16 shows that $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ has comparable normalized performance with respect to that of LPG. The results show that $\text{SGPlan}_\text{t}(\text{MIPS}, 20)$ is able to improve over LPG in 49.6% of the cases in quality or 54.2% of the cases in time. The

[4] At the time of this revision, we have finished implementing $\text{SGPlan}_\text{g}(\text{FF}/\text{LPG}, N)$, 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]. $\text{SGPlan}_\text{g}(\text{FF}/\text{LPG}, N)$ 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.



(a)



(b)

Fig. 15. Normalized time and quality of $\text{SGPlan}_t(\text{MIPS}, 20)$ 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 $(1, 1)$. (a) Distribution of the quality of solutions found by $\text{SGPlan}_t(\text{MIPS}, 20)$ normalized with respect to that of the corresponding solutions of MIPS, each using the same amount of time taken by MIPS to find the solution. (b) Distribution of the times taken by $\text{SGPlan}_t(\text{MIPS}, 20)$ to find a solution of the same or better quality as that of MIPS, normalized with respect to the time taken by MIPS to find the solution.

improvements over LPG are not substantial because $\text{SGPlan}_t(\text{MIPS}, 20)$ 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 $\text{SGPlan}_t(\text{MIPS}, 20)$ 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 $\text{SGPlan}_t(\text{MIPS}, 20)$ and LPG. The results show that $\text{SGPlan}_t(\text{MIPS}, 20)$ outperforms MIPS in most of the instances tested, and that $\text{SGPlan}_t(\text{MIPS}, 20)$ has comparable performance as LPG.

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

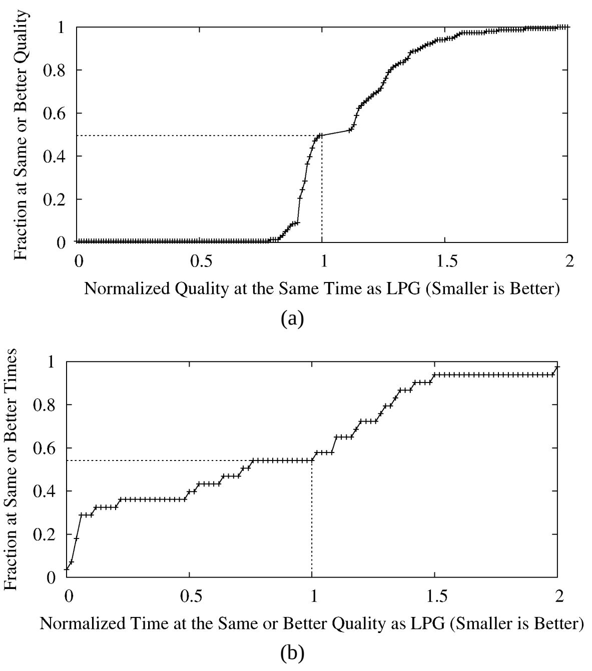


Fig. 16. Normalized time and quality of $\text{SGPlan}_t(\text{MIPS}, 20)$ with respect to LPG on the 154 instances solvable by LPG (out of 160). The time and quality of LPG are normalized to $(1, 1)$. (a) Distribution of the quality of solutions found by $\text{SGPlan}_t(\text{MIPS}, 20)$ normalized with respect to that of the corresponding solutions of LPG, each using the same amount of time taken by LPG to find the solution. (b) Distribution of the times taken by $\text{SGPlan}_t(\text{MIPS}, 20)$ to find a solution of the same or better quality as that of LPG, normalized with respect to the time taken by LPG to find the solution.

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

Table 1
Results on MIPS, SGPlan$_t$(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 1000 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$_t$(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$_t$(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$_t$(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

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f	$Time_1$	Sol_1	$Time_f$	Sol_f
DepotNumericl	149	32	20	22.6	20	$\boxed{22.6}$	30	32.6	40	22.6
DepotNumeric2	398	43	40	37.4	90	$\boxed{33.9}$	40	53.9	250	33.9
DepotNumericS	2543	31	100	23	10080	$\boxed{11}$	370	41	76820	12
DepotNumeric4	–	–	1930	54.6	37540	$\boxed{30}$	1080	214.8	$\underline{32980}$	30
DepotNumeric5	–	–	30340	129.5	144300	$\boxed{75.2}$	7310	331.401	239360	94.6
DepotNumeric6	–	–	–	–	–	–	–	–	–	–
DepotNumeric7	1522	38	160	96.1	380	$\boxed{36.3}$	250	186.2	3190	36.3
DepotNumeric8	1605	28	2180	35	390350	$\boxed{14}$	410	25	$\underline{36460}$	14
DepotNumeric9	–	–	–	–	–	–	264980	413.7	264980	$\underline{413.7}$
DepotNumeric10	107850	16	140	15	77660	$\boxed{10}$	250	19	178740	10
DepotSim1	38	46.12	30	45	40020	$\boxed{24}$	40	28	$\underline{70}$	24
DepotSim2	51	73.2	30	52	1040	$\boxed{31}$	50	48	$\underline{850}$	31
DepotSim3	476	103.39	620	129	278960	$\boxed{40}$	1850	110	317880	40
DepotSim4	135485	130.03	290	67	15070	$\boxed{28}$	1480	153	63630	28
DepotSim5	–	–	24400	157	24400	$\boxed{157}$	1270	192	567290	$\underline{142}$
DepotSim6	–	–	–	–	–	–	33010	292	177590	$\underline{225}$
DepotSim7	169	67.27	1080	46	62450	$\boxed{27}$	100	63	26130	29
DepotSim8	302327	111.039	530	70	190010	$\boxed{43}$	870	93	402240	46
DepotSim9	–	–	39680	254	384400	$\boxed{139}$	32970	203	349010	172
DepotSim10	164389	91.04	130	82	67520	$\boxed{38}$	160	71	113620	38
DepotSim11	–	–	2780	194	259880	$\boxed{84}$	6790	267	228010	97
DepotSim12	–	–	–	–	–	–	425800	280	425800	$\underline{280}$
DepotSim13	450	84.025	350	175	1820	$\boxed{42}$	190	82	27510	$\underline{41}$
DepotSim14	–	–	470	56	76550	$\boxed{43}$	530	114	90190	$\underline{42}$

(*continued on next page*)

Table 1 (*continued*)

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f	$Time_1$	Sol_1	$Time_f$	Sol_f
DepotSim15	–	–	159490	262	576510	$\boxed{198}$	200370	346	359850	$\underline{170}$
DepotSim16	439830	73.032	200	59	5950	$\boxed{29}$	240	71	3060	$\underline{28}$
DepotTime1	40	59.4811	40	45.625	40	$\boxed{45.625}$	30	44.635	40	$\underline{44.635}$
DepotTime2	54	92.3111	40	50	12210	$\boxed{36.667}$	100	63.667	$\underline{590}$	37.667
DepotTime3	623	255.082	2120	548.908	548910	$\boxed{56.508}$	230	156.9	174260	59.683
DepotTime4	143242	199.456	180	78	93210	$\boxed{74.25}$	1920	147.499	384170	$\underline{74}$
DepotTime5	–	–	58520	556.905	58520	$\boxed{556.905}$	150720	2049.384	204330	677.595
DepotTime6	–	–	–	–	–	–	–	–	–	–
DepotTime7	134	149.791	280	95.714	280	$\boxed{95.714}$	600	691.577	13610	$\underline{94.714}$
DepotTime8	754	136.313	3300	76.286	505390	$\boxed{37.709}$	620	73.905	372910	$\underline{35.905}$
DepotTime9	–	–	11371	1083.33	292220	$\boxed{1038.5}$	47390	1294.999	591380	$\underline{914.001}$
DepotTime10	172322	245.57	130	159.416	437720	$\boxed{109.749}$	140	207.417	241320	111
DepotTime11	–	–	31090	535.334	432840	$\boxed{437.443}$	7600	630.139	575110	447.028
DepotTime12	–	–	334500	194.096	334500	$\boxed{194.096}$	–	–	–	–
DepotTime13	1203	104.526	430	190.55	32460	$\boxed{50.042}$	210	88.095	567390	$\underline{47.961}$
DepotTime14	–	–	870	370.199	330520	$\boxed{122}$	640	297.032	263450	122.2
DepotTime15	–	–	242110	268.596	242110	$\boxed{268.596}$	153770	760.95	153770	760.95
DepotTime16	429824	84.8653	230	44.889	55870	$\boxed{12.001}$	310	55.501	67850	13.555
DepotTime17	–	–	1720	97.05	608270	$\boxed{39.125}$	2030	141.2	273450	$\underline{36.125}$
DepotTime18	–	–	74090	320.292	289510	$\boxed{244.276}$	15470	379.016	15470	379.016
DepotTime19	–	–	580	330.286	172440	$\boxed{152.867}$	500	240.838	398940	$\underline{150.4}$
DepotTime20	–	–	175940	814.294	261470	$\boxed{507.68}$	96160	682.233	310060	$\underline{424.705}$
DriveLogNumeric1	90	1099	20	777.2	42920	$\boxed{777.198}$	20	777.2	42730	777.199
DriveLogNumeric2	89	1497	30	2006.5	324250	$\boxed{979.998}$	40	1472.2	3980	979.999
DriveLogNumeric3	92	907	20	1213.6	6340	$\boxed{641.9}$	40	1175.7	133780	$\underline{637.998}$
DriveLogNumeric4	112	715	30	1038.9	4590	$\boxed{706.9}$	40	933	15210	$\underline{704}$
DriveLogNumeric5	124	878	30	1335.5	247570	$\boxed{581.3}$	30	797.1	63640	581.8
DriveLogNumeric6	130.1	1667	30	968.4	52150	$\boxed{968.396}$	30	1000.5	37960	$\underline{965.498}$
DriveLogNumeric7	123.1	$\boxed{866}$	30	978.7	131630	870.698	40	1191	88960	$\underline{866.799}$
DriveLogNumeric8	19680	3273	30	2135.6	31380	$\boxed{1430.2}$	40	3493.5	176900	1430.299

(*continued on next page*)

Table 1 (*continued*)

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$
DriveLogNumeric9	629	3002	40	3239	383640	$\boxed{1816.5}$	50	2376.4	80510	1834.2
DriveLogNumeric10	278	402	50	346.3	346800	$\boxed{144.3}$	50	403.4	182240	$\underline{143.4}$
DriveLogNumeric11	7250	616	60	573.2	151870	$\boxed{340.5}$	90	1234.1	528370	354.4
DriveLogNumeric12	14320	3227	230	4972.6	7430	$\boxed{2033.6}$	280	5067.401	496720	2119.6
DriveLogNumeric13	2521	2148	160	1969.4	500690	$\boxed{1161.3}$	170	2575.3	148980	1223.3
DriveLogNumeric14	34433.1	3347	1230	10692.5	89360	$\boxed{1752.4}$	300	5092.502	575720	$\underline{1677.9}$
DriveLogNumeric15	12421	1753	570	1568	489580	$\boxed{1157.1}$	720	3254.2	239360	1227.1
DriveLogNumeric16	–	–	74893	14398.3	74893	$\boxed{14398.3}$	109730	15932.389	109730	15932.389
DriveLogNumeric17	–	–	19510	20583.4	580450	$\boxed{7384.7}$	4440	11671.297	146520	8816.497
DriveLogNumeric18	–	–	11160	12480.1	250850	$\boxed{9055}$	7980	11240.894	387730	$\underline{8402.803}$
DriveLogNumeric19	–	–	605000	25219.6	605000	$\boxed{25219.6}$	204360	27835.066	535560	$\underline{24096.869}$
DriveLogNumeric20	–	–	484120	19323.2	564590	$\boxed{15084}$	86530	15059.186	547320	$\underline{13163.992}$
DriveLogSim1	90	92.07	30	92	10030	$\boxed{91}$	20	91	$\underline{20}$	91
DriveLogSim2	90	92.21	30	103	40	$\boxed{92}$	30	130	80	92
DriveLogSim3	98	$\boxed{40.07}$	20	47	40	40.1	30	47	50	$\underline{40}$
DriveLogSim4	99	89.16	30	98	2180	$\boxed{52}$	30	98	$\underline{490}$	52
DriveLogSim5	112	51.19	30	119	10740	$\boxed{51}$	30	101	$\underline{70}$	51
DriveLogSim6	117	64.13	30	94	31030	$\boxed{52}$	40	101	$\underline{70}$	52
DriveLogSim7	122	40.09	30	51	70	$\boxed{40}$	30	113	330	40
DriveLogSim8	279.1	111.26	40	134	239310	$\boxed{52}$	40	130	$\underline{17240}$	52
DriveLogSim9	202	264.31	40	192	81840	$\boxed{92}$	40	222	$\underline{65760}$	92
DriveLogSim10	269.1	61.21	40	50	680	$\boxed{38}$	60	113	3390	38
DriveLogSim11	351	99.21	50	85	442350	$\boxed{67}$	50	105	433490	$\underline{65}$
DriveLogSim12	1772	252.41	400	578	477470	$\boxed{168}$	410	748	228440	$\underline{156}$
DriveLogSim13	1734	104.29	130	258	62250	$\boxed{102}$	140	462	70980	102
DriveLogSim14	2403	226.44	1910	1562	315350	$\boxed{106}$	170	290	$\underline{130570}$	109
DriveLogSim15	13620	265.43	700	311	333100	$\boxed{125}$	610	665	422000	$\underline{113}$
DriveLogSim16	–	–	–	–	–	–	–	–	–	–
DriveLogSim17	549119	$\boxed{223.94}$	4350	867	346820	245	8270	389	384210	$\underline{238}$
DriveLogSim18	–	–	65920	672	208150	$\boxed{327}$	4330	747	308990	327

(*continued on next page*)

Table 1 (*continued*)

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$
DriveLogSim19	–	–	58080	1030	58080	$\boxed{1030}$	599250	3460	599250	3460
DriveLogSim20	–	–	178550	771	311250	$\boxed{293}$	73400	713	522800	375
DriveLogTime1	65	303	30	303	10020	$\boxed{302}$	20	303	$\underline{40}$	302
DriveLogTime2	80	310	30	321	66340	$\boxed{245}$	40	409	97700	246
DriveLogTime3	75	$\boxed{173}$	30	213	40	$\boxed{173}$	30	173	$\underline{30}$	173
DriveLogTime4	75	392	30	339	7170	$\boxed{230}$	30	498	60480	230
DriveLogTime5	103	112	30	330	14570	$\boxed{102}$	30	268	$\underline{2790}$	102
DriveLogTime6	124	260	40	242	31050	$\boxed{168}$	30	168	$\underline{30}$	168
DriveLogTime7	123	268	30	321	219520	$\boxed{200}$	40	421	$\underline{170860}$	200
DriveLogTime8	235	313	50	346	9200	$\boxed{206}$	40	261	54940	$\underline{202}$
DriveLogTime9	233	980	40	695	13150	$\boxed{320}$	40	714	42940	$\underline{318}$
DriveLogTime10	287	340	50	442	62070	$\boxed{93}$	50	193	$\underline{4240}$	93
DriveLogTime11	343	391	50	501	21060	$\boxed{232}$	50	271	$\underline{210}$	232
DriveLogTime12	1530	611	260	2072	32970	$\boxed{319}$	250	1555	463590	327
DriveLogTime13	1256	558	120	651	22410	$\boxed{388}$	140	1052	$\underline{5460}$	388
DriveLogTime14	2303	1049	260	1066	286970	$\boxed{287}$	260	564	7800	328
DriveLogTime15	9853	893	970	671	153350	$\boxed{242}$	420	661	486120	265
DriveLogTime16	–	–	–	–	–	–	–	–	–	–
DriveLogTime17	236244	$\boxed{954.94}$	3590	2508	247890	983	1940	875	496970	$\underline{563}$
DriveLogTime18	–	–	32470	1752	540870	$\boxed{1026}$	15050	3613	322340	1061
DriveLogTime19	–	–	–	–	–	–	135920	4705	439960	$\underline{1888}$
DriveLogTime20	–	–	94730	1721	442220	$\boxed{1528}$	35590	3264	445890	$\underline{1436}$
ZenoTravelNumeric1	72	13564	16	13564	124	$\boxed{13563.9}$	20	13564	150	13563.957
ZenoTravelNumeric2	70	$\boxed{6786}$	50	18130.7	5100	6786.19	30	9770.399	306840	6786.283
ZenoTravelNumeric3	92	7505	30	11014.6	110050	$\boxed{4505.38}$	30	6756.5	438010	4505.479
ZenoTravelNumeric4	91	16964	30	19968.5	295510	$\boxed{16960.5}$	30	37037.805	136020	16960.553
ZenoTravelNumeric5	100	19916	30	13013.9	342970	$\boxed{3974.79}$	30	26043.695	523460	$\underline{3973.879}$
ZenoTravelNumeric6	112	35282	40	122595	107230	$\boxed{15206}$	40	30475.098	482710	$\underline{15204.996}$
ZenoTravelNumeric7	103.1	16472	30	14435	471810	$\boxed{8257.9}$	40	14434.998	590740	$\underline{7276.897}$
ZenoTravelNumeric8	183	33543	50	53577.9	433040	$\boxed{18682.9}$	40	33613	103100	$\underline{18682.852}$

(*continued on next page*)

Table 1 (*continued*)

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$	*Time*$_1$	*Sol*$_1$	*Time*$_f$	*Sol*$_f$
ZenoTravelNumeric9	192	28047	70	20806.1	376820	$\boxed{4787.8}$	70	14469.096	324990	$\underline{4743.602}$
ZenoTravelNumeric10	214.1	79564	70	100002	532000	$\boxed{40341.8}$	100	146918.594	293390	$\underline{40337.891}$
ZenoTravelNumeric11	252	55480	250	134844	248740	$\boxed{16694.9}$	330	137610	232520	$\underline{13389.096}$
ZenoTravelNumeric12	306	41310	120	65678.4	240580	$\boxed{21793.6}$	110	72017.117	175780	$\underline{20665.695}$
ZenoTravelNumeric13	413	82230	140	132594	532050	$\boxed{20435.3}$	140	91871.719	569060	23555.893
ZenoTravelNumeric14	6247	233381	1200	230617	224180	$\boxed{147478}$	1250	273628.688	256040	$\underline{136158.406}$
ZenoTravelNumeric15	15890	147618	3090	152790	160850	$\boxed{68264.9}$	5110	216508.531	554260	79056.133
ZenoTravelNumeric16	31652	143282	12920	129253	227350	$\boxed{80313.6}$	9720	146591.344	164940	$\underline{67332.422}$
ZenoTravelNumeric17	64438	$\boxed{182558}$	302440	213325	582570	204608	27490	250293.875	78670	$\underline{186907.75}$
ZenoTravelNumeric18	123543.4	$\boxed{70794}$	498710	161821	498710	161821	54010	148765.719	485910	$\underline{83159.25}$
ZenoTravelNumeric19	135935	212997	594600	328254	717330	$\boxed{205481}$	70660	185678.062	107460	$\underline{168739.938}$
ZenoTravelNumeric20	245335	$\boxed{89937}$	1054220	602522	1054220	602522	214660	612788.875	434000	$\underline{410776.656}$
ZenoTravelSim1	80	180.01	1	180.08	10	$\boxed{173}$	20	180	260	173
ZenoTravelSim2	78	643.06	20	998	174720	$\boxed{592}$	40	643	$\underline{9970}$	592
ZenoTravelSim3	1431	683.09	30	1052	30340	$\boxed{280}$	40	649	$\underline{180}$	280
ZenoTravelSim4	124	936.11	30	1272	78640	$\boxed{622}$	50	882	98140	$\underline{522}$
ZenoTravelSim5	234	690.13	80	2686	14210	$\boxed{400}$	50	656	$\underline{610}$	400
ZenoTravelSim6	330.1	480.12	40	826	8440	$\boxed{323}$	50	935	144160	323
ZenoTravelSim7	213	716.16	90	1501	208670	$\boxed{692}$	80	1482	71550	$\underline{679}$
ZenoTravelSim8	1243	846.13	130	939	475840	$\boxed{549}$	160	1173	541200	$\underline{529}$
ZenoTravelSim9	1376	1256.24	260	2005	9880	$\boxed{556}$	270	2486	77490	$\underline{536}$
ZenoTravelSim10	1523	1432.29	260	2253	313860	$\boxed{643}$	150	882	548370	$\underline{490}$
ZenoTravelSim11	3734.1	1219.19	240	1768	68030	$\boxed{430}$	200	1039	130330	$\underline{423}$
ZenoTravelSim12	3551	1179.29	410	2834	523970	$\boxed{643}$	380	1442	133280	$\underline{576}$
ZenoTravelSim13	3603	913.31	300	2510	174150	$\boxed{643}$	1590	5565	110650	$\underline{636}$
ZenoTravelSim14	124518.4	1099.36	5790	1722	52140	$\boxed{883}$	3280	1535	511800	$\underline{756}$
ZenoTravelSim15	233530	1758.4	29730	1898	400300	$\boxed{989}$	11680	2108	390190	1042
ZenoTravelSim16	–	–	22870	2207	284310	$\boxed{1476}$	29170	3067	679820	$\underline{1233}$
ZenoTravelSim17	–	–	772460	5330	874770	$\boxed{4801}$	126230	3874	253060	$\underline{3843}$
ZenoTravelSim18	–	–	103426	3401.4	103426	$\boxed{3401.4}$	198170	5693	675610	4544

(*continued on next page*)

Table 1 (*continued*)

Problem ID	MIPS		SGPlan$_t$(MIPS, 20)				LPG			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f	$Time_1$	Sol_1	$Time_f$	Sol_f
ZenoTravelSim19	–	–	–	–	–	–	270380	4055	270380	$\underline{4055}$
ZenoTravelSim20	–	–	–	–	–	–	–	–	–	–
ZenoTravelTime1	50	27.257	10	27.256	10	$\boxed{27.256}$	20	27.256	20	27.256
ZenoTravelTime2	50	$\boxed{30.2104}$	20	30.4096	20	30.4096	20	30.51	20	30.51
ZenoTravelTimeS	78	18.1527	30	32.4213	21030	$\boxed{17.56}$	20	25.080	9780	$\underline{17.443}$
ZenoTravelTime4	82	153.294	40	230.33	21090	$\boxed{74.3}$	30	162.096	2640	75
ZenoTravelTimeS	99	37.7473	30	21.864	40	$\boxed{18.271}$	30	24.291	60	19.071
ZenoTravelTime6	93	51.7826	40	65.982	8080	$\boxed{40.5}$	40	80.84	174070	41.173
ZenoTravelTime7	112	142.179	40	114.203	159410	$\boxed{86.695}$	30	178.96	157260	$\underline{85.295}$
ZenoTravelTime8	201	160.639	110	237.676	441350	$\boxed{132.738}$	50	167.735	16180	$\underline{125.317}$
ZenoTravelTime9	223	119.82	90	113.862	264220	$\boxed{54.887}$	80	126.458	455660	58.089
ZenoTravelTime10	221	181.68	90	244.51	490090	$\boxed{126.756}$	70	426.288	510560	$\underline{121.712}$
ZenoTravelTime11	276	155.308	110	171.481	69320	$\boxed{111.517}$	90	172.094	445880	$\underline{106.87}$
ZenoTravelTime12	353	126.007	130	218.589	545780	$\boxed{75.895}$	80	189.528	182900	84.895
ZenoTravelTime13	455	90.28	110	128.758	245110	$\boxed{56.9374}$	110	153.593	81300	56.994
ZenoTravelTime14	7823	$\boxed{375.056}$	1880	739.127	412240	394.88	1290	588.987	112650	$\underline{342.651}$

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.

Acknowledgements

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

Appendix A. Proof of Theorem 1

The proof consists of two parts.

"($\Rightarrow$)" part: Given x^*, we need to prove that there exist finite $\alpha^{**} > \alpha^* \geqslant 0$ and $\beta^{**} > \beta^* \geqslant 0$ that satisfy (13). The first inequality in (13) is true for all α and β 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 $\alpha^* \geqslant 0$ and $\beta^* \geqslant 0$ such that the inequality is satisfied for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. Let $x = x^* + \varepsilon \vec{p}$, where $\|\vec{p}\| = 1$ is a unit directional vector and ε 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 $f(x) \geqslant f(x^*)$ and, regardless the choice of the penalties,

$$L_c(x, \alpha^{**}, \beta^{**}) = f(x) \geqslant f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \tag{A.1}$$

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

$$\left|h_k(x)\right| > \xi > 0 = h_k(x^*). \tag{A.2}$$

The above must be true because $h_k(x)$ would be continuous along $\vec{p}$ if (A.2) were false.

If we set $\alpha_k^{**} > \alpha_k^* = 1$ and when ε is small enough, then from (A.2):

$$\begin{aligned} L_c(x, \alpha^{**}, \beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**} \left|h_i(x)\right| + \sum_{j=1}^{r} \beta_j^{**} \max\bigl(0, g_j(x)\bigr) \\ &\geqslant f(x) + \alpha_k^{**} \left|h_k(x)\right| > f(x^*) + \varepsilon \nabla_x f(x^*)^{\mathrm{T}} \vec{p} + \mathrm{o}(\varepsilon^2) + \alpha_k^* \xi \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \end{aligned} \tag{A.3}$$

(3) If there exists an active inequality constraint function g_k that is discontinuous along $\vec{p}$, then for a small enough ε, there exists a finite positive ξ such that $\max(0, g_k(x)) > \xi > 0$. If we set $\beta_k^{**} > \beta_k^* = 1$ and when ε is small enough, this condition implies that:

$$\begin{aligned} L_c(x,\alpha^{**},\beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**}\big|h_i(x)\big| + \sum_{j=1}^{r} \beta_j^{**} \max\big(0, g_j(x)\big) \\ &\geqslant f(x) + \beta_k^{**} \max\big(0, g_k(x)\big) \\ &> f(x^*) + \varepsilon \nabla_x f(x^*)^{\mathrm{T}} \vec{p} + \mathrm{o}(\varepsilon^2) + \beta_k^* \xi \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \end{aligned} \tag{A.4}$$

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

$$\begin{aligned} L_c(x,\alpha^{**},\beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**}\big|h_i(x)\big| + \sum_{j=1}^{r} \beta_j^{**} \max\big(0, g_j(x)\big) \\ &\geqslant f(x) + \alpha_k^{**}\big|h_k(x)\big| \\ &\geqslant f(x^*) + \varepsilon \nabla_x f(x^*)^{\mathrm{T}} \vec{p} + \mathrm{o}(\varepsilon^2) + \alpha_k^{**}\varepsilon\big|D_x(h_k(x^*), \vec{p})\big| \\ &\geqslant f(x^*) + \varepsilon\big(\alpha_k^{**}\big|D_x\big(h_k(x^*), \vec{p}\big)\big| - \big|\nabla_x f(x^*)^{\mathrm{T}} \vec{p}\big|\big) + \mathrm{o}(\varepsilon^2) \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \end{aligned} \tag{A.5}$$

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

"($\Leftarrow$)" part: Assuming (13) is satisfied, we need to prove that x^* is a CLM_c. Point x^* is feasible because the first inequality in (13) can only be satisfied when $h(x^*) = 0$ and $g(x^*) \leqslant 0$. 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_c.

Appendix B. Proof of Theorem 2

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

"($\Rightarrow$)" part: Given y^*, we need to prove that there exist finite $\alpha^{**} > \alpha^* \geqslant 0$ and $\beta^{**} > \beta^* \geqslant 0$ that satisfy (16). In order for α^* and β^* to exist for every CLM_d y^*, α^* and β^* 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 $|h(y)| > 0$ (*respectively* $\max(0, g(y)) > 0$), there is at least one constraint that is not satisfied. For each such constraint, we update its penalty as follows:

$$\alpha_i^* \leftarrow \max\left\{\alpha_i^*, \frac{f(y^*) - f(y)}{|h_i(y)|}\right\} \quad \text{if } \big|h_i(y)\big| > 0, \tag{B.1}$$

$$\beta_j^* \leftarrow \max\left\{\beta_j^*, \frac{f(y^*) - f(y)}{\max(0, g_j(y))}\right\} \quad \text{if } \max\bigl(0, g_j(y)\bigr) > 0. \tag{B.2}$$

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 α^* and β^* values.

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

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 $g(y) \leqslant 0$ that:

$$L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) \leqslant f(y) = L_d(y, \alpha^{**}, \beta^{**}). \tag{B.3}$$

For all $y \in \mathcal{N}_d(y^*)$ such that $h(y) \neq 0$ (*respectively* $g(y) \nleqslant 0$), there must exist at least one constraint that is not satisfied. From (B.1) and (B.2), we know for this constraint that:

$$\begin{aligned}
&\alpha_i^{**} > \alpha_i^* \geqslant \frac{f(y^*) - f(y)}{|h_i(y)|} \\
&\quad \Rightarrow L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) < f(y) + \alpha_i^{**}\bigl|h_i(y)\bigr| \quad \text{if } \bigl|h_i(y)\bigr| > 0, && \text{(B.4)} \\
&\beta_j^{**} > \beta_j^* \geqslant \frac{f(y^*) - f(y)}{\max(0, g_j(y))} \\
&\quad \Rightarrow L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) < f(y) + \beta_j^{**} \max\bigl(0, g_j(y)\bigr) \\
&\qquad \text{if } \max\bigl(0, g_j(y)\bigr) > 0. && \text{(B.5)}
\end{aligned}$$

Further, since $\sum_{k=1,k\neq i}^{m} \alpha_k^* |h_k(y)| \geqslant 0$ (*respectively* $\sum_{k=1,k\neq j}^{r} \beta_k^* \max(0, g_j(y)) \geqslant 0$), it is clear that:

$$\begin{aligned}
L_d(y^*, \alpha^{**}, \beta^{**}) = f(y^*) &\leqslant f(y) + \sum_{i=1}^{m} \alpha_i^{**}\bigl|h_i(y)\bigr| + \sum_{j=1}^{r} \beta_i^{**} \max\bigl(0, g_j(y)\bigr) \\
&= L_d(y, \alpha^{**}, \beta^{**}).
\end{aligned}$$

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

"($\Leftarrow$)" 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 $\alpha^* \geqslant 0$ and $\beta^* \geqslant 0$ so that $(x^*, y^*, \alpha^{**}, \beta^{**})$ satisfies (20). The first inequality in (20) is true for all α and β, 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 α_c^* and β_c^* such that:

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leqslant L_m(x, y^*, \alpha^{**}, \beta^{**}), \quad \forall x \in \mathcal{N}_c(x^*)|_{y^*},\ \alpha^{**} > \alpha_c^* \geqslant 0,\ \text{and } \beta^{**} > \beta_c^* \geqslant 0. \tag{C.1}$$

Similarly, fixing x at x^* converts P_m into P_d. Hence, from Theorem 2, we know that there exist finite α_d^* and β_d^* such that, for the same α^{**} and β^{**} in (C.1):

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leqslant L_m(x^*, y, \alpha^{**}, \beta^{**}), \quad \forall y \in \mathcal{N}_d(y^*)|_{x^*},\ \alpha^{**} > \alpha_d^* \geqslant 0,\ \text{and } \beta^{**} > \beta_d^* \geqslant 0. \tag{C.2}$$

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

$$\alpha^* = \max(\alpha_c^*, \alpha_d^*) = \left(\max(\alpha_{c_1}^*, \alpha_{d_1}^*), \ldots, \max(\alpha_{c_m}^*, \alpha_{d_m}^*)\right)^{\mathrm{T}}, \tag{C.3}$$

$$\beta^* = \max(\beta_c^*, \beta_d^*) = \left(\max(\beta_{c_1}^*, \beta_{d_1}^*), \ldots, \max(\beta_{c_r}^*, \beta_{d_r}^*)\right)^{\mathrm{T}}, \tag{C.4}$$

we conclude, based on (C.1) and (C.2), that the second inequality in (20) is satisfied for all $(x, y) \in \mathcal{N}_m(x^*, y^*)$ and for any $\alpha^{**} > \alpha^* \geqslant 0$ and $\beta^{**} > \beta^* \geqslant 0$.

"($\Leftarrow$)" 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.

Appendix D. Proof of Theorem 5

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

"($\Rightarrow$)" 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^*.

"($\Leftarrow$)" 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:

$$L_m(z^*, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}) \nleqslant L_m(z, \alpha^{**}, \beta^{**}, \gamma^{**}, \eta^{**}). \tag{D.1}$$

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).

References

[1] E. Aarts, J. Korst, Simulated Annealing and Boltzmann Machines, J. Wiley and Sons, New York, 1989.
[2] M. Avriel, Nonlinear Programming: Analysis and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1976.
[3] R. Bellman, S. Dreyfus, Applied Dynamic Programming, Princeton Univ. Press, Princeton, NJ, 1962.

[4] D.P. Bertsekas, Nonlinear Programming, Athena Scientific, Belmont, MA, 1999.
[5] A.L. Blum, M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1997) 281–300.
[6] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (1) (2001).
[7] Y.X. Chen, C.-W. Hsu, B.W. Wah, SGPlan: Subgoal partitioning and resolution in planning, in: Proc. Fourth Internat. Planning Competition, Internat. Conf. on Automated Planning and Scheduling, 2004.
[8] Y.X. Chen, B.W. Wah, Automated planning and scheduling using calculus of variations in discrete space, in: Proc. Internat. Conf. on Automated Planning and Scheduling, 2003, pp. 2–11.
[9] S. Chien, et al., ASPEN—Automating space mission operations using automated planning and scheduling, in: Proc. SpaceOps, Space Operations Organization, 2000.
[10] P. Doherty, J. Kvarnstrom, TALplanner: An empirical investigation of a temporal logic-based forward chaining planner, in: Proc. Sixth Internat. Workshop on Temporal Logic-based Forward Chaining Planner, AIPS, 1999, pp. 47–54.
[11] M.A. Duran, I.E. Grossmann, An outer approximation algorithm for a class of mixed-integer nonlinear programs, Math. Programming 36 (1986) 306–307.
[12] S. Edelkamp, Mixed propositional and numerical planning in the model checking integrated planning system, in: Proc. Workshop on Planning in Temporal Domains, AIPS, 2002, pp. 47–55.
[13] M.P. Fourman, Propositional planning, in: Proc. Workshop on Model Theoretic Approaches to Planning, AIPS, 2000.
[14] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, J. Artificial Intelligence Res. 20 (2003) 61–124.
[15] A.M. Geoffrion, Generalized Benders decomposition, J. Optim. Theory Appl. 10 (4) (1972) 237–241.
[16] A. Gerevini, I. Serina, LPG: A planner based on local search for planning graphs with action costs, in: Proc. of the Sixth Internat. Conf. on AI Planning and Scheduling, Morgan Kaufman, Santa Mateo, CA, 2002, pp. 12–22.
[17] N.I.M. Gould, D. Orban, Ph.L. Toint, An interior-point L1-penalty method for nonlinear optimization, Technical Report, RAL-TR-2003-022, Rutherford Appleton Laboratory Chilton, Oxfordshire, UK, 2003.
[18] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, J. Artificial Intelligence Res. 14 (2001) 253–302.
[19] K. Holmberg, On the convergence of the cross decomposition, Math. Programming 47 (1990) 269–316.
[20] K. Holmberg, Generalized cross decomposition applied to nonlinear integer programming problems: duality gaps and convexification in parts, Optimization 23 (1992) 341–364.
[21] A.K. Jónsson, P.H. Morris, N. Muscettola, K. Rajan, Planning in interplanetary space: Theory and practice, in: Proc. 2nd Internat. NASA Workshop on Planning and Scheduling for Space, NASA, 2000.
[22] H. Kautz, B. Selman, Pushing the envelope: planning, propositional logic, and stochastic search, in: Proc. 13th National Conference on Artificial Intelligence, AAAI, 1996, pp. 1194–1201.
[23] H. Kautz, B. Selman, Unifying SAT-based and graph-based planning, in: Proc. Internat. Joint Conf. on Artificial Intelligence, IJCAI, 1999.
[24] H. Kautz, J.P. Walser, Integer optimization models of AI planning problems, Knowledge Engrg. Rev. 15 (1) (2000) 101–117.
[25] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[26] H.W. Kuhn, A.W. Tucker, Nonlinear programming, in: Proc. Second Berkeley Symp. Math. Stat. Prob., University of California Press, 1951, pp. 481–492.
[27] F. Lin, A planner called R, AI Magazine (2001) 73–76.
[28] D. Long, M. Fox, Efficient implementation of the plan graph in STAN, J. Artificial Intelligence Res. 10 (1999) 87–115.
[29] D.G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, Reading, MA, 1984.
[30] D. Nau, H. Muoz-Avila, Y. Cao, A. Lotem, S. Mitchell, Total-order planning with partially ordered subtasks, in: Proc. Internat. Joint Conf. on Artificial Intelligence, IJCAI, 2001, pp. 425–430.
[31] R.S. Nigenda, X. Nguyen, S. Kambhampati, AltAlt: Combining the advantages of Graphplan and heuristic state search, Technical Report, Arizona State University, 2000.
[32] J. Penberethy, D. Weld, UCPOP: A sound, complete, partial order planner for ADL, in: Proc. 3rd Internat. Conf. on Principles of Knowledge Representation and Reasoning, KR Inc., 1992, pp. 103–114.

[33] J. Penberethy, D. Weld, Temporal planning with continuous change, in: Proc. 12th National Conf. on AI, AAAI, 1994, pp. 1010–1015.

[34] G. Rabideau, S. Chien, C. Eggemeyer, T. Mann, J. Willis, S. Siewert, P. Stone, Interactive, repair-based planning and scheduling for shuttle payload operations, in: Proc. Aerospace Conf., IEEE, 1997, pp. 325–341.

[35] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, Iterative repair planning for spacecraft operations in the ASPEN system, in: Proc. Internat. Symp. on Artificial Intelligence Robotics and Automation in Space, European Space Agency, 1999.

[36] R.L. Rardin, Optimization in Operations Research, Prentice-Hall, Englewood Cliffs, NJ, 1998.

[37] I. Refanidis, I. Vlahavas, The GRT planner, AI Magazine (2001) 63–66.

[38] I. Refanidis, I. Vlahavas, The MO-GRT system: Heuristic planning with multiple criteria, in: Proc. Workshop on Planning and Scheduling with Multiple Criteria, AIPS, 2002.

[39] T.J. Van Roy, Cross decomposition for mixed integer programming, Math. Programming 25 (1983) 46–63.

[40] H.S. Ryoo, N.V. Sahinidis, A branch-and-reduce approach to global optimization, J. Global Optim. 8 (2) (1996) 107–139.

[41] N.V. Sahinidis, BARON: A general purpose global optimization software package, J. Global Optim. 8 (2) (1996) 201–205.

[42] D. Schuurmans, F. Southey, Local search characteristics of incomplete sat procedures, Artificial Intelligence 132 (2) (2001) 121–150.

[43] B. Selman, H.A. Kautz, An empirical study of greedy local search for satisfiability testing, in: Proc. of 11th National Conf. on Artificial Intelligence, AAAI, 1993, pp. 46–51.

[44] Y. Shang, B.W. Wah, A discrete Lagrangian based global search method for solving satisfiability problems, J. Global Optim. 12 (1) (1998) 61–99.

[45] M.B.D. Subbarao, S. Kambhampati, Sapa: A domain-independent heuristic metric temporal planner, Technical Report, Arizona State University, 2002.

[46] A. Tate, B. Drabble, R. Kirby, O-Plan2: an open architecture for command, planning and control, in: Intelligent Scheduling, Morgan Kaufmann, 1994, pp. 213–239.

[47] B.W. Wah, Y.X. Chen, Partitioning of temporal planning problems in mixed space using the theory of extended saddle points, in: Proc. IEEE Internat. Conf. on Tools with Artificial Intelligence, 2003, pp. 266–273.

[48] B.W. Wah, Y.X. Chen, Subgoal partitioning and global search for solving temporal planning problems in mixed space, Internat. J. Artificial Intelligence Tools 13 (4) (2004) 767–790.

[49] B.W. Wah, Y.X. Chen, Solving large-scale nonlinear programming problems by constraint partitioning, in: Proc. Principles and Practice of Constraint Programming, Springer-Verlag, Berlin, 2005.

[50] B.W. Wah, T. Wang, Simulated annealing with asymptotic convergence for nonlinear constrained global optimization, in: Proc. Principles and Practice of Constraint Programming, Springer-Verlag, Berlin, 1999, pp. 461–475.

[51] B.W. Wah, Z. Wu, The theory of discrete Lagrange multipliers for nonlinear discrete optimization, in: Proc. Principles and Practice of Constraint Programming, Springer-Verlag, Berlin, 1999, pp. 28–42.

[52] D. Wilkins, Can AI planners solve practical problems?, Computational Intelligence (1990) 232–246.

[53] J. Willis, G. Rabideau, C. Wilklow, The citizen explorer scheduling system, in: Proc. Aerospace Conf., IEEE, 1999.

[54] S. Wolfman, D. Weld, Combining linear programming and satisfiability solving for resource planning, Knowledge Engrg. Rev. 15 (1) (2000).

[55] Z. Wu, The theory and applications of nonlinear constrained optimization using Lagrange multipliers, Ph.D. Thesis, Department of Computer Science, Univ. of Illinois, Urbana, IL, 2001.