International Journal on Artificial Intelligence Tools
Vol. 13, No. 4 (2004) 767–790

World Scientific
www.worldscientific.com

SUBGOAL PARTITIONING AND GLOBAL SEARCH FOR SOLVING TEMPORAL PLANNING PROBLEMS IN MIXED SPACE*

BENJAMIN W. WAH

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

YIXIN CHEN

Department of Computer Science
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801
United States of America
chen@manip.crhc.uiuc.edu

Received 3 November 2003
Revised 27 February 2004
Accepted 1 May 2004

We study in this paper the partitioning of the constraints of a temporal planning problem by subgoals, their sequential evaluation before parallelizing the actions, and the resolution of inconsistent global constraints across subgoals. Using an ℓ_1-penalty formulation and the theory of extended saddle points, we propose a global-search strategy that looks for local minima in the original-variable space of the ℓ_1-penalty function and for local maxima in the penalty space. Our approach improves over a previous scheme that partitions constraints along the temporal horizon. The previous scheme leads to many global constraints that relate states in adjacent stages, which means that an incorrect assignment of states in an earlier stage of the horizon may violate a global constraint in a later stage of the horizon. To resolve the violated global constraint in this case, state changes will need to propagate sequentially through multiple stages, often leading to a search that gets stuck in an infeasible point for an extended period of time. In this paper, we propose to partition all the constraints by subgoals and to add new global constraints in order to ensure that state assignments of a subgoal are consistent with those in other subgoals. Such an approach allows the information on incorrect state assignments in one subgoal to propagate quickly to other subgoals. Using MIPS as the basic planner

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

in a partitioned implementation, we demonstrate significant improvements in time and quality in solving some PDDL2.1 benchmark problems.

Keywords: Extended saddle point; global search; mixed-integer nonlinear programming problem; nonlinear constraint; partitioning; subgoal; temporal planning.

1. Introduction

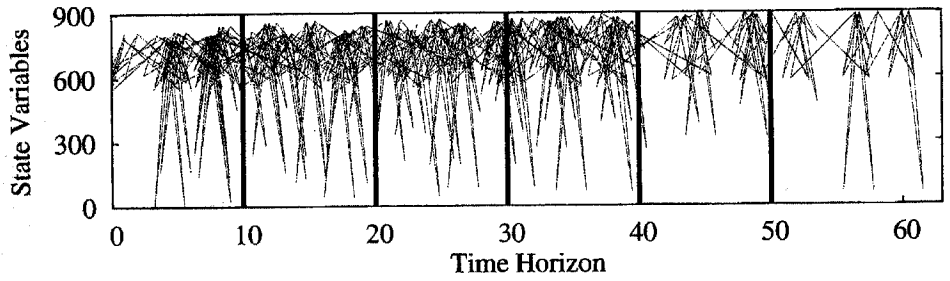
A temporal planning problem involves arranging actions and assigning resources order to accomplish a given set of tasks over a period of time and to optimize one more objectives. It can be defined loosely by a set of states whose variables may discrete, continuous, or mixed; a discrete or continuous temporal horizon; a set actions defining valid transitions between states; a set of effects to be evaluated each state or action; a set of constraints to be satisfied in each state or througho an action; and a set of goals to be achieved.[28]

Our goal in this paper is to study the partitioning of temporal planning pro lems and effective global search strategies for finding locally optimal feasible plan In our approach, we formulate a planning problem as a *mixed-integer* (involving di crete and continuous variables) *nonlinear programming* (MINLP) problem. Base on the subgoals of a planning problem, we partition those constraints related t a subgoal into a subset (called a *stage*). We identify a *local constraint* to involv state variables related to a subgoal in one stage and a *global constraint* to involv state variables across two or more stages. Using formal mathematical condition that govern constrained local minima, we develop efficient search algorithms fo resolving unsatisfied local and global constraints and for optimizing objectives.

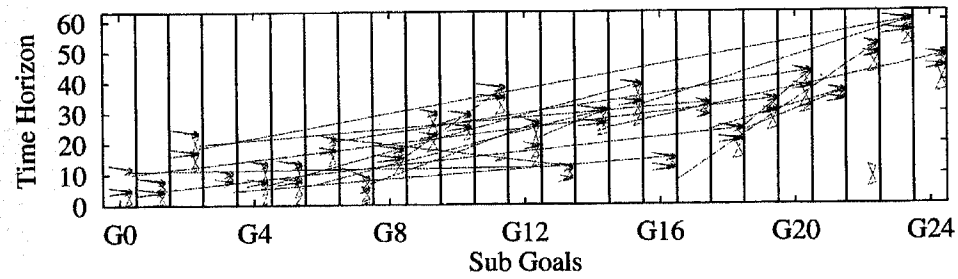
The success of our approach depends on the ability to partition the constraint of a large planning problem into subproblems in such a way that each can be solvec easily and that global constraints relating subproblems can be resolved quickly. Although many of the constraints in a temporal planning problem are related to activities and events with temporal localities and can be partitioned in such a way that a majority of the constraints are temporally local, such a partitioning does not always lead to the most efficient evaluation.

As an example, Figure 1a shows the 720 constraints of an initial (infeasible) schedule generated by MIPS[9] in solving *zenoTraveTimeNumeric20*.[11] The problem involves transporting people in planes, using the fast and slow modes of movement. The duration of a move is computed from its distance and speed. By partitioning the horizon into six stages,[31] there are 642 local constraints and 78 global constraints.

We have found that the partitioning of constraints in PDDL2.1 benchmarks along the temporal horizon often leads to many global constraints that only relate states in adjacent stages (as illustrated in Figure 1a). As a result, when a violated subgoal is caused by an incorrect assignment of states in an early stage of the horizon, the change of the incorrect assignments will have to propagate sequentially through multiple stages. Oftentimes, the propagation of such information may lead to a search getting stuck in an infeasible point for an extended period of time.[31] To



a) Partitioning of constraints by temporal horizon



b) Partitioning of constraints by subgoals

Fig. 1. Two ways to partition the constraints in an initial infeasible schedule generated by MIPS in solving *zenoTraveTimeNumeric20*. In (a), each of the 720 constraint is shown as a line that relates two states (labeled in the y-axis) scheduled at two times in the horizon (x-axis). The partitioning of the horizon into six stages (separated by bold vertical lines) leads to 642 local constraints and 78 global constraints. In (b), the 720 constraints are partitioned into 24 stages according to the 24 subgoals. Each constraint, after parallelizing the actions, is shown as a line that relates two states in a subgoal (x-axis) scheduled at two times in the horizon (y-axis), where each stage includes all the states in the schedule. In addition, $\frac{24\times 23}{2}$ binary global constraints are added, each enforcing the consistency of the state assignments in one subgoal to those of another. The figure only shows the 21 violated global constraints.

address this issue, we propose in this paper to partition the constraints according to subgoals (see Figure 1b), evaluate the subgoals sequentially, resolve any inconsistent state assignments among them, and parallelize their actions. New global constraints are added to ensure that state assignments of all subgoals are consistent.

Our approach partitions all the constraints of a planning problem into $N+1$ stages, where stage t, $t = 0, \ldots, N$, has local *state vector* $z(t) = (z_1(t), \ldots, z_{u_t}(t))^T$ of u_t mixed variables, m_t local equality constraints, and r_t local inequality constraints. Here, $z(t)$ includes all variables that appear in any of the local constraints in stage t. Since the partitioning is by constraints, the $N+1$ state vectors $z(0), \ldots, z(N)$ may overlap with each other. The MINLP formulation of the partitioned problem is as follows:

$$
\begin{array}{lllll}
(P_t): & \min\limits_{z} & J(z) & & \\
& \text{subject to} & h^{(t)}(z(t)) = 0, & g^{(t)}(z(t)) \leq 0, & \text{(local constraints)} \\
& \text{and} & H(z) = 0, & G(z) \leq 0. & \text{(global constraints)}
\end{array} \quad (1)
$$

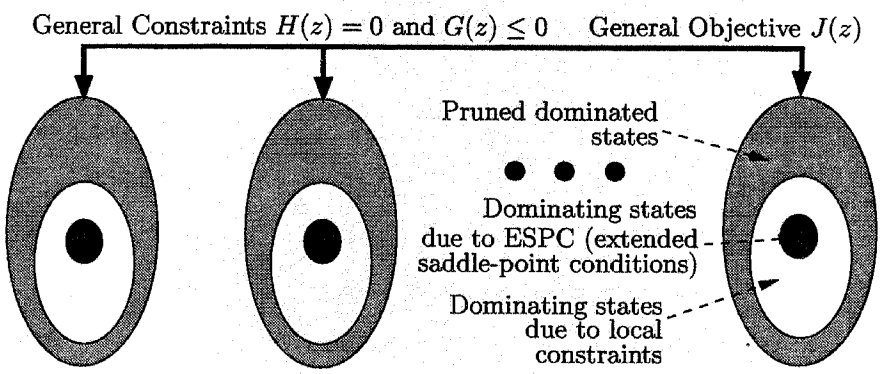


Fig. 2. The pruning of states that do not satisfy local constraints and ESPC in each stage lea to a significant reduction in the joint search space for resolving violated global constraints acr the stages.

Here, $h^{(t)} = (h_1^{(t)}, \ldots, h_{m_t}^{(t)})^T$ and $g^{(t)} = (g_1^{(t)}, \ldots, g_{r_t}^{(t)})^T$ are local-constraint fun tions in stage t that involve $z(t)$; and $H = (H_1, \ldots, H_p)^T$ and $G = (G_1, \ldots, G_q)$ are global-constraint functions that involve $z \in Z$, the variables of the origin problem. We assume that J is continuous and differentiable with respect to its con tinuous variables, that f is lower bounded, and that g and h are general function that are not necessarily continuous or differentiable and that can be unbounded. A solution to (1) is a *plan* that consists of an assignment of z.

The partitioning of constraints allows us to divide a large problem into smalle subproblems, solve each independently, and resolve those violated global constraints afterwards. One of the major benefits of our approach is that each partitioned subproblem is much easier to solve than the original problem because it involves a substantially smaller number of constraints. Further, the joint search space across all the subproblems in which violated global constraints must be resolved is reduced dramatically because it is made up of subspaces that must satisfy the local constraints in each subproblem (the first inner ellipse in each stage of Figure 2). In this paper, we propose new conditions that allow the search space of each subproblem to be further reduced (the second inner ellipse in each stage of Figure 2) before resolving violated global constraints.

In addition to reducing the problem complexity, another benefit of constraint partitioning is that it leads to smaller subproblems of a similar nature. As a result, existing solvers can be employed to solve these subproblems with little or no modification. Without the need to develop new solvers for each subproblem, search techniques in existing solvers can be employed. Further, new and better solvers developed in the future can be integrated easily in our approach.

The multi-stage problem in (1) cannot be solved by dynamic programming because its states in different stages may overlap, and 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.

The problem formulated cannot be solved by existing penalty-based methods because they have no effective way for resolving violated global constraints after solving the partitioned subproblems. Without resolving the violated local and global constraints together, these methods will have to rely on expensive trial and error to find the correct penalty for each global constraint after solving the subproblems.

For a similar reason, existing planners do not exploit constraint partitioning. Existing AI planning and scheduling methods can be classified based on their state and temporal representations and the search techniques used.

a) *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. Examples include UCPOP,[24] Graphplan,[4] STAN,[21] PropPLAN,[10] and System R.[20] Systematic solvers explore a search space by partitioning it into subspaces and by exploring each as a complete planning problem. They are not amenable to constraint partitioning because they have no means for resolving inconsistent global constraints after solving the subproblems.

Local searches employ heuristic guidance functions to search in discrete path space. Examples include HSP,[5] FF,[14] AltAlt,[23] GRT,[27] and ASPEN.[7] Similar to systematic searches, these heuristic solvers explore a partitioned subspace represented as a complete planning problem and employ guidance heuristics that are evaluated over the entire temporal horizon in order to estimate the distance from a state to the goal state. They do not have means to resolve inconsistent global constraints when subproblems are partitioned by constraints.

Last, transformation methods convert a problem into a constrained optimization or satisfaction problem before solving it by existing solvers. Examples include SATPLAN,[16] Blackbox,[17] and ILP-PLAN.[18] Transformation methods are not amenable to constraint partitioning because they rely on SAT and ILP solvers that do not support such partitioning.

b) *Discrete-time mixed-state methods* employ systematic searches, heuristic searches, and transformation methods. Examples include SIPE-2,[33] O-Plan2,[30] Metric-FF,[14] GRT-R,[27] and LPSAT.[34] The search methods employed by these planners are not amenable to constraint partitioning for reasons similar to those in discrete-time discrete-state methods.

c) *Continuous-time mixed-state methods* can be classified into systematic, heuristic, and local searches. Examples include LPG,[12] MIPS,[9] Sapa,[29] ZENO,[25] SHOP2,[22] TALplanner,[8] and Europa.[15] For reasons similar to those in discrete-time discrete-state methods, the methods in these planners do not have means to resolve inconsistent global constraints when subproblems are partitioned by constraints.

In the next section, we review existing mathematical programming techniques. In Section 3, we present the necessary and sufficient extended saddle-point condition (ESPC) that governs the correctness of algorithms for solving (1). Since ESPC allows (1) to be solved under an extensive range of penalties for each constraint, it simplifies the resolution of inconsistent global constraints across partitioned sub-

problems. Moreover, the application of ESPC in each subproblem leads tc reduction in its search space, which limits the search space in which glo straints need to be evaluated. In Section 4, we show some global-search s in the ℓ_1-penalty-function space in order to help a search escape from infea cal minima. Finally, we present in Section 5 an application of the search st on the MIPS planner and the solution of some PDDL2.1 planning benchm

The results in this paper extend our previous work on variable partiti discrete space and time.[6] Our previous work is based on the partitioning of v of a discrete planning problem into disjoint subsets. It can be considered a case of constraint partitioning in which the variable sets after partitioning disjoint. The ESPC presented in this paper are also more general becau applicable to continuous and mixed problems as well as to discrete problen

2. Mathematical Programming Background

Consider the following *continuous nonlinear programming* (CNLP) proble continuous and differentiable f, $h = (h_1, \ldots, h_m)^T$, and $g = (g_1, \ldots, g_r)^T$ in real space:

$$(P_c): \qquad \min_x \ f(x) \ \text{ where } x = (x_1, \ldots, x_v)^T \in \mathcal{R}^v$$
$$\text{subject to } \ h(x) = 0 \ \text{ and } \ g(x) \leq 0.$$

The goal of solving P_c is to find a constrained local minimum x^* with r to $\mathcal{N}_c(x^*) = \{x' : \|x' - x^*\| \leq \epsilon \text{ and } \epsilon \to 0\}$, the *continuous neighborhood* of

Definition 2.1. Point x^* is a CLM_c, a constrained local minimum with r to the continuous neighborhood of x^*, of P_c if x^* is feasible and $f(x^*) \leq f($ all feasible $x \in \mathcal{N}_c(x^*)$.

Based on Lagrange-multiplier vectors $\lambda = (\lambda_1, \ldots, \lambda_m)^T \in \mathcal{R}^m$ and $(\mu_1, \ldots, \mu_r)^T \in \mathcal{R}^r$, the Lagrangian function of P_c is defined as:

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

a) *Karush-Kuhn-Tucker (KKT) necessary condition.*[3] Assuming x^* is a C and a regular point,[a] then there exist unique λ^* and μ^* such that:

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

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

The unique λ and μ that satisfy (4) can be found by solving a system of linear equations in λ, μ, and x or by iterative procedures. The latter approac

[a]Point x is a *regular point* if gradient vectors of equality constraints $\nabla h_1(x), \ldots, \nabla h_m(x)$ active inequality constraints $\nabla g_{a_1}(x), \ldots, \nabla g_{a_l}(x), a_i \in A(x)$ (the set of active constraints) linearly independent.

taken in existing sequential quadratic programming (SQP) solvers, such as SNOPT and LANCELOT. For instance, SNOPT solves the system of nonlinear equations iteratively by first forming a quadratic approximation, solving the quadratic model, and updating estimates of x, λ, and μ, until unique x, λ, and μ are found. Since, in general, the system of equations in (4) must be solved together, inconsistent assignments across subproblems cannot be resolved easily when each partitioned subproblem is solved beforehand.

A recent approach called the interior-point ℓ_1-penalty method[13] does not require finding unique λ and μ. However, the approach is limited to solving CNLPs with continuous and differentiable functions and without partitioning, and cannot be applied to solve partitioned MINLP planning problems studied in this paper.

b) *Sufficient saddle-point condition.*[2] The concept of saddle points has been studied extensively in the past. Here, x^* is a saddle point of P_c if there exist unique λ^* and μ^* such that:

$$L(x^*, \lambda, \mu) \leq L(x^*, \lambda^*, \mu^*) \leq L(x, \lambda^*, \mu^*) \tag{5}$$

for all x that satisfies $\|x - x^*\| < \epsilon$ and all $\lambda \in \mathcal{R}^m$ and $\mu \in \mathcal{R}^r$. This condition is only sufficient but not necessary because there may not exist feasible λ^* and μ^* that satisfy (5) for each x^*.

In practice, (5) is not used to find unique x, λ, and μ that satisfy (4) because it is difficult to solve for unique λ^* and μ^* using a system of nonlinear inequalities.

c) *Penalty formulations.* A penalty function is the summation of the objective and the constraint functions weighted by penalties. In a penalty formulation, the goal is to find suitable penalties in such a way that the x that minimizes the penalty function corresponds to the CLM_c of P_c. In general, the minimum of a penalty function is only necessary but not sufficient to be a CLM_c of the constrained model because suitable penalties may not exist. Unless the penalties are chosen properly, the minimization of a penalty function does not always lead to a CLM_c.

Stronger necessary and sufficient conditions also exist for penalty formulations. A *static-penalty approach*[3,26] transforms P_c into the following unconstrained minimization problem:

$$L_\rho(x, \gamma, \psi) = f(x) + \gamma^T |h(x)|^\rho + \psi^T \max(0, g(x))^\rho, \tag{6}$$

where $\rho > 0$. By choosing $\rho = 1$, there exist finite and sufficiently large penalty vectors $\gamma \in \mathcal{R}^m$ and $\psi \in \mathcal{R}^r$ such that x^*, a global minimum of $L_\rho(x, \gamma, \psi)$, corresponds to a *constrained global minimum* (CGM_c) of P_c. Although such penalties always exist, there is no systematic method for choosing them.

To overcome the difficulty of finding γ in (6), a *dynamic-penalty approach*[26] increases penalties gradually and solves for the optimal solution of a sequence of unconstrained problems. Although it is easier to apply than a static-penalty approach, it does not guarantee a feasible solution eventually when each unconstrained problem is solved suboptimally.

In contrast to methods for solving continuous problems, MINLP methods gene ally decompose the search space (rather than the constraints) of a MINLP into sul problems in such a way that, after fixing a subset of the variables, each subproble is convex and easily solvable, or can be relaxed and approximated. There are sev eral types of these algorithms, including generalized Benders decomposition, outc approximation, generalized cross decomposition, and branch-and-reduce methods All those methods require the functions of subproblems to be convex or factorable which is a condition difficult to meet in planning problems.

In short, existing theory based on KKT and saddle points applies only to CNLP and generally requires the solution of unique Lagrange multipliers. The theory doe not apply in solving MINLPs because unique Lagrange multipliers may not exis for each MINLP solution. In the next section, we present a new theory of extendec saddle points that does not require finding unique penalties and can be applied tc solve partitioned MINLPs.

3. Theory of Extended Saddle Points

Given planning problem (1), we describe our theory of extended saddle points in mixed space based on an ℓ_1-penalty function. We show a necessary and sufficient condition that is satisfied for a large range of penalty values and the decomposition of the necessary and sufficient condition for partitioned problems.

3.1. *Extended saddle-point condition (ESPC) for mixed optimization*

Consider the following MINLP:

$$(P_m): \quad \min_{x,y} \quad f(x,y), \quad x \in \mathcal{R}^v \text{ and } y \in \mathcal{D}^w \qquad (7)$$
$$\text{subject to} \quad h(x,y) = 0 \text{ and } g(x,y) \leq 0,$$

where f is continuous and differentiable with respect to x, and $g = (g_1, \ldots, g_r)^T$ and $h = (h_1, \ldots, h_m)^T$ are general functions that are not necessarily continuous or differentiable. We further assume that f is lower bounded, while g and h can be unbounded.

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^*). To define $\mathcal{N}_m(x, y)$, we need to specify its continuous and discrete counterparts. Although a continuous neighborhood is well defined, there is no accepted definition of a discrete neighborhood. We define it as follows:

Definition 3.1. A user-defined *discrete neighborhood*[1] $\mathcal{N}_d(y)$ of $y \in \mathcal{D}^w$ is a *finite* user-defined set of points $\{y' \in \mathcal{D}^w\}$, where y' is reachable from y in one step, $y' \in \mathcal{N}_d(y) \Longleftrightarrow y \in \mathcal{N}_d(y')$, and every y'' can be reached from any y in one or more steps through neighboring points.

Intuitively, $\mathcal{N}_d(y)$ represents points that are perturbed from y, with no requirement that there be valid state transitions from y. Next, we define a mixed neighborhood and a constrained local minimum in this neighborhood:

Definition 3.2. A user-defined *mixed neighborhood* $\mathcal{N}_m(x, y)$ in mixed space $\mathcal{R}^v \times \mathcal{D}^w$ is:

$$\mathcal{N}_m(x,y) = \left\{ (x', y) \;\middle|\; x' \in \mathcal{N}_c(x) \right\} \cup \left\{ (x, y') \;\middle|\; y' \in \mathcal{N}_d(y) \right\}. \tag{8}$$

Definition 3.3. Point (x^*, y^*) is a CLM_m (a *constrained local minimum in a mixed neighborhood*) of P_m if (x^*, y^*) is feasible and $f(x^*, y^*) \leq f(x, y)$ for all feasible $(x, y) \in \mathcal{N}_m(x^*, y^*)$.

There are two distinct features of CLM_m. First, the set of CLM_m of a problem is neighborhood dependent because it depends on the user-defined discrete neighborhood; that is, (x, y) may be CLM_m with respect to $\mathcal{N}_m(x, y)$ but may not be with respect to $\mathcal{N}'_m(x, y)$. Although the choice of neighborhoods does not affect the validity of a search as long as a consistent definition is used throughout, it may affect the time to find a CLM_m. Second, a discrete neighborhood has a *finite* number of points. As a result, the verification of a point to be CLM_m with respect to its discrete neighborhood can be done by comparing its objective value against those of the *finite* number of discrete neighboring points. This feature allows the search of a descent direction in discrete neighborhood to be done by enumeration or greedy search, rather than by differentiation.

Next, we state the following two concepts used in our theory.

Definition 3.4. The ℓ_1-penalty function of P_m in (7) is defined as follows:

$$L_m(x, y, \alpha, \beta) = f(x, y) + \alpha^T |h(x, y)| + \beta^T \max(0, g(x, y)), \tag{9}$$

where $\alpha \in \mathcal{R}^m$ and $\beta \in \mathcal{R}^r$ are penalty vectors.

Definition 3.5. $D_x(f(x', y'), \vec{p})$, the *subdifferential* of function f at $(x', y') \in X \times Y$ along direction $\vec{p} \in X$ in the x subspace, represents the rate of change of $f(x', y')$ under an infinitely small perturbation along $\vec{p}$. That is,

$$D_x(f(x', y'), \vec{p}) = \lim_{\epsilon \to 0} \frac{f(x' + \epsilon\vec{p}, y') - f(x', y')}{\epsilon}. \tag{10}$$

Since we define our mixed neighborhood to be the union of points perturbed in either the discrete or the continuous subspace, but not both, we can develop our theory for the two subspaces separately. In the continuous subspace, we need the following constraint qualification condition in order to rule out the special case in which all continuous constraints have zero subdifferential along a direction. A similar concept is not needed in the discrete subspace because constraint functions are not changing continuously there.

Definition 3.6. *Constraint qualification for ESPC.* Solution $(x^*, y^*) \in X \times Y$ of P_m meets the constraint qualification if there exists no direction $\vec{p} \in X$ along which the subdifferentials of continuous equality and continuous active inequality constraints are all zero. That is,

$$\nexists\, \vec{p} \in X \text{ such that } D_x(h_i(x^*, y^*), \vec{p}) = 0 \text{ and } D_x(g_j(x^*, y^*), \vec{p}) = 0 \quad \text{for all } i \in C_h \text{ and } j \in C_g,$$

where C_h and C_g are, respectively, the sets of indices of continuous equality and continuous active inequality constraints.

The intuitive meaning of constraint qualification can be explained as follows. Consider a feasible point (x', y') and a nearby infeasible neighboring point $(x' + \vec{p}, y')$, where the objective function f at (x', y') decreases along $\vec{p}$ and all active constraints at (x', y') have zero subdifferentials along $\vec{p}$. In this case, it is not possible to find finite penalty values that penalize the violated constraints at $(x' + \vec{p}, y')$ in order to have a local minimum of the ℓ_1-penalty function at (x', y') with respect to $(x' + \vec{p}, y')$. In short, if the above scenario is true for any direction $\vec{p}$ at (x', y'), then there does not exist finite penalty values that lead to a local minimum of the penalty function at (x', y').

Our constraint-qualification condition requires the subdifferential of at least one active constraints to be non-zero along all directions $\vec{p}$ in the x subspace. For CNLPs, the condition rules out the case in which there exists a direction $\vec{p}$ along which all active constraints are continuous and have zero subdifferentials. Our condition is different from the regularity condition in KKT in that, it requires at least one of the continuous constraints to have non-zero subdifferential, whereas the regularity condition requires the gradients of constraint functions to be all non-zero and linearly independent. Our condition is less restricted than the regularity condition because we can penalize an infeasible point in our ℓ_1-penalty function using only one (rather than all) violated constraint.

Next, we state our main theorem.

Theorem 3.1. *Necessary and sufficient ESPC on CLM_m of P_m.* Suppose $(x^*, y^*) \in \mathcal{R}^v \times \mathcal{D}^w$ of P_m satisfies the constraint qualification condition, then (x^*, y^*) is a CLM_m of P_m if and only if there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that the following is satisfied:

$$L_m(x^*, y^*, \alpha, \beta) \leq L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y, \alpha^{**}, \beta^{**}) \quad \text{where } \alpha^{**} > \alpha^* \text{ and } \beta^{**} > \beta^* \tag{11}$$

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

The proof consists of three parts. The first part proves that ESPC is necessary and sufficient for continuous problems. The necessity proof starts from the KKT condition, applies a Taylor-series expansion of the ℓ_1-penalty function around x^*, and proves the inequalities in (11). The sufficiency proof is done by construction.

The second part of the proof for ESPC of discrete problems is extended from our previous work.[32] Finally, the proof of ESPC for mixed problems is based on the definition of mixed neighborhoods in Definition 3.2, which allows continuous and discrete subspaces to be considered separately. We omit the details of the proof due to space limitation.

The following corollary facilitates the implementation of (11) and is stated without proof. It follows directly from Definition 3.2 on $\mathcal{N}_m(x, y)$, which allows (11) to be partitioned into two independent necessary conditions. It allows thresholds of penalties to be found in the discrete and continuous subspaces separately, and the maximum values taken to be the final thresholds in mixed space. Note that such partitioning cannot be accomplished if a mixed neighborhood based on the Cartesian product of $\mathcal{N}_c(x)$ and $\mathcal{N}_d(y)$ were used.

Corollary 3.1. Given $\mathcal{N}_m(x, y)$, ESPC in (11) can be rewritten into two necessary conditions that, collectively, are sufficient:

$$L_m(x^*, y^*, \alpha, \beta) \leq L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x^*, y, \alpha^{**}, \beta^{**}) \quad (12)$$

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y^*, \alpha^{**}, \beta^{**}) \quad (13)$$

where $y \in \mathcal{N}_d(y^* \mid \text{for given } x^*)$ and $x \in \mathcal{N}_c(x^* \mid \text{for given } y^*)$.

3.2. *ESPC for partitioned problems*

Based on the results in the last section, we can now solve P_t in (1) by partitioning it into subproblems. We first show that plan z, a CLM_m with respect to its mixed neighborhood $\mathcal{N}_m(z)$, satisfies the ESPC in Theorem 3.1. To solve (1) efficiently, we define a mixed neighborhood for partitioned problems and decompose the ESPC in (11) into a set of necessary conditions that collectively are sufficient. The partitioned conditions can then be implemented by finding local saddle points in each stage of P_t and by resolving the unsatisfied global constraints using appropriate penalties.

To simplify our discussion, we do not partition $z(t)$ in stage t into discrete and continuous parts in the following derivation, although it is understood that each stage will need to be further decomposed in the same way as in (8). To enable the partitioning of the ESPC into independent necessary conditions, we define $\mathcal{N}_p(z)$, the neighborhood of z for a partitioned problem, as follows.

Definition 3.7. $\mathcal{N}_p(z)$, the *mixed neighborhood* of z for a partitioned problem, is:

$$\mathcal{N}_p(z) = \bigcup_{t=0}^{N} \mathcal{N}_p^{(t)}(z) = \bigcup_{t=0}^{N} \left\{ z' \;\middle|\; z'(t) \in \mathcal{N}_m(z(t)) \text{ and } \forall z_i \notin z(t),\ z_i' = z_i \right\}, \quad (14)$$

where $\mathcal{N}_m(z(t))$ is the mixed neighborhood of variable vector $z(t)$ in stage t.

Intuitively, $\mathcal{N}_p(z)$ is separated into $N + 1$ neighborhoods, each perturbing z in one of the stages of P_t, while keeping the overlapped variables consistent across

multiple stages. The size of $\mathcal{N}_p(z)$ defined in (14) is smaller than the Cartesian product of the neighborhoods across all stages.

By considering P_t as an MINLP and by defining the corresponding ℓ_1-penalty function, we can apply Theorem 3.1 as follows.

Definition 3.8. The ℓ_1-penalty function for P_t in (1) is:

$$L_m(z,\alpha,\beta,\gamma,\eta) = J(z) + \sum_{t=0}^{N}\left\{\alpha(t)^T|h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t))\right\} + \gamma^T|H(z)| + \eta^T \max(0, G(z)), \quad (15)$$

where $\alpha(t) = (\alpha_1(t),\ldots,\alpha_{m_t}(t))^T \in \mathcal{R}^{m_t}$ and $\beta(t) = (\beta_1(t),\ldots,\beta_{r_t}(t))^T \in \mathcal{R}^{r_t}$ are vectors of penalties for the local constraints in stage t, and $\gamma = (\gamma_1,\ldots,\gamma_p)^T \in \mathcal{R}^p$ and $\eta = (\eta_1,\ldots,\eta_q)^T \in \mathcal{R}^q$ are vectors of penalties for the global constraints.

Lemma 3.1. Assuming z^* of P_t satisfies the constraint qualification condition in Definition 3.6, then z^* is a CLM_m of (1) with respect to $\mathcal{N}_p(z)$ if and only if there exist finite nonnegative α^*, β^*, γ^* and η^* such that the following condition is satisfied:

$$L_m(z^*,\alpha,\beta,\gamma,\eta) \le L_m(z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}) \le L_m(z,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}), \quad (16)$$
$$\text{where } \alpha^{**} > \alpha^*, \beta^{**} > \beta^*, \gamma^{**} > \gamma^* \text{ and } \eta^{**} > \eta^*$$

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

Next, we show that (16) can be partitioned into a set of necessary conditions that collectively are sufficient.

Theorem 3.2. *Partitioned necessary and sufficient ESPC on CLM_m of P_t.* Given $\mathcal{N}_p(z)$, ESPC in (16) can be rewritten into $N+2$ necessary conditions that collectively are sufficient:

$$\Gamma_m^{(t)}(z^*,\alpha(t),\beta(t),\gamma^{**},\eta^{**}) \le \Gamma_m^{(t)}(z^*,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**}) \le \Gamma_m^{(t)}(z,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**}), \quad (17)$$

$$L_m(z^*,\alpha^{**},\beta^{**},\gamma,\eta) \le L_m(z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}), \quad (18)$$

for all $z \in \mathcal{N}_p^{(t)}(z^*)$, $\alpha(t) \in \mathcal{R}^{m_t}$, $\beta(t) \in \mathcal{R}^{r_t}$, $\gamma \in \mathcal{R}^p$, and $\eta \in \mathcal{R}^q$, where $t = 0,\ldots,N$ and

$$\Gamma_m^{(t)}(z,\alpha(t),\beta(t),\gamma,\eta) = J(z) + \alpha(t)^T|h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t))) + \gamma^T|H(z)| + \eta^T \max(0, G(z)). \quad (19)$$

Theorem 3.2 shows that the original ESPC in Theorem 3.1 can be partitioned into multiple necessary conditions, each of which corresponds to finding an extended saddle point in a stage. With fixed γ and η, we are actually finding $z(t)$ that solves

the following MINLP in stage t whose objective is biased by the global constraints:

$$\min_{z(t)} \; J(z) + \gamma^T H(z) + \eta^T G(z) \tag{20}$$
$$\text{subject to } \; h(t, z(t)) = 0 \quad \text{and} \quad g(t, z(t)) \leq 0.$$

As a result, the solution of the original problem is now reduced to solving multiple smaller subproblems. The bias due to the global constraints is important because it provides better guidance in solving the subproblem in stage t.

4. Global Search Implementing ESPC

An important aspect of Theorem 3.1 over the original saddle-point condition in (5) is that, instead of solving a system of nonlinear equations to find unique λ^* and μ^* that minimize $L(x, \lambda^*, \mu^*)$ at x^*, it suffices to find any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. Such a property allows the solution of P_m to be implemented iteratively by looking for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$ in an outer loop, and for a local minimum (x^*, y^*) of $L_m(x, y, \alpha, \beta)$ with respect to points in $\mathcal{N}_m(x^*, y^*)$ in an inner loop.

Figure 3a shows the pseudo code implementing the conditions in Corollary 3.1. The two inner loops look for local minima of $L_m(x, y, \alpha, \beta)$ in the continuous and discrete neighborhoods, whereas the outer loop performs ascents on α and β for unsatisfied global constraints. The algorithm ends when a CLM_m has been found.

The iterative search can be extended to the partitioned conditions in Theorem 3.2. One approach is to solve (20) in stage t directly as a planning problem. Since this is a well-defined MINLP, any existing solver with little modification can be used. We have studied this approach in discrete planning domains by using ASPEN to solve subproblems partitioned by a discrete version of Theorem 3.2.[6]

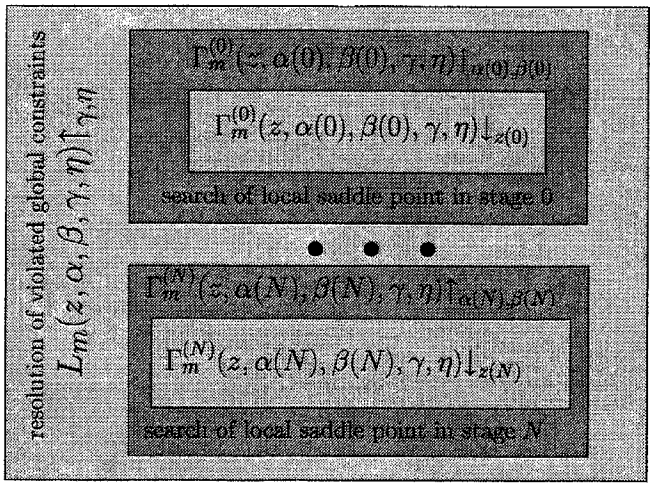
A more general approach for solving (20) is to look for a local saddle point of $\Gamma_m^{(t)}(z, \alpha(t), \beta(t), \gamma, \eta)$ that satisfies (17), using fixed γ and η associated with the global constraints. The process is shown in the two inner nested loops in Figure 3b. After performing the local searches, the penalties on unsatisfied global constraints are increased in the outer loop. The search iterates until a constrained local minimum has been found.

Because our proposed approach does not require a unique penalty value for each global constraint, we can separate their updates from those of z and implement the search iteratively. Such an approach cannot be used when the traditional Lagrangian theory is applied. In the traditional theory, each global constraint must be associated with a unique Lagrange-multiplier value when the search converges. Without resolving all the local constraints and the global constraints together, it will be difficult for any iterative search to converge to a unique Lagrange-multiplier value for each global constraint.

A search based on our iterative approach may get stuck in an infeasible region when the objective is too small or when the penalties and/or constraint violations are too large. In this case, increasing the penalties will further deepen the infeasible region, making it impossible for a descent algorithm to escape from this region.

$\alpha \longrightarrow 0$; $\beta \longrightarrow 0$;
repeat
  increase α_i by δ_i if $h_i(x,y) \neq 0$ for all i;
  increase β_j by δ_j if $g_j(x,y) \not\leq 0$ for all j;
  repeat
    perform descent of $L_m(x,y,\alpha,\beta)$ with respect to x for given y;
  until a local minimum of $L_m(x,y,\alpha,\beta)$ with respect
    to x for given y has been found;
  repeat
    perform descent of $L_m(x,y,\alpha,\beta)$ with respect to y for given x;
  until a local minimum of $L_m(x,y,\alpha,\beta)$ with respect
    to y for given x has been found;
until a CLM_m of P_m has been found or ($\alpha > \bar{\alpha}^*$ and $\beta > \bar{\beta}^*$);

a) Implementation of Corollary 3.1



b) Implementation of Theorem 3.2

Fig. 3. Iterative implementation of ESPC to look for CLM_m of P_m and that of partitioned ESPC to look for CLM_m of P_t.

To address this issue, we can change either the ascent algorithm in the two outer loops of Figure 3b or its descent algorithm in the innermost loops. The ascent algorithm can be changed to allow increases as well as decreases of penalties α, β, γ, and η. The goal of decreases is to "lower" the barrier in the penalty function in order for local descents in the innermost loops to escape from an infeasible region. For the same reason as in dynamic penalty methods, α, β, γ, and η should be increased gradually in order to help the search escape from local minima of $L_m(x,y,\alpha,\beta,\gamma,\eta)$. Once α, β, γ, and η reach their maximum thresholds, they can be scaled down, and the search is repeated.

In a similar way, the descent algorithm in the innermost loops can be changed to allow descents as well as ascents. Descent algorithms used in temporal planning problems can get stuck in infeasible local minima easily because functions in planning problems may not be in closed form and their exact gradients are not available. To cope with this issue, probes generated may be accepted based on stochastic

criteria. For example, the descent algorithm in our partitioned implementation of ASPEN[6] accepts probes with larger penalty values according to the Metropolis probability in order to allow occasional ascents. In degenerate cases, restarts may be needed in order to escape from deep infeasible regions.

In this paper, we only implement the first strategy, namely, the periodic decreases of penalties in addition to ascents in the ℓ_1-penalty-function space with respect to the penalties. It is not necessary to implement both strategies because they offset each other in their effects.

Yet another strategy that helps identify promising regions to explore is to relax the constraints initially and to tighten them gradually as feasible solutions to the relaxed problem have been found. The approach allows potentially promising starting points to be found, at a cost much lower than that of solving the original problem. If a feasible local minimum is not found after the constraints have been tightened, the constraints can be relaxed again in order to allow the search to move to a different region in the search space. By relaxing and tightening the constraints repeatedly, a search can move from one region to another. We plan to study this strategy in the future.

5. Partitioned Implementation of MIPS

In this section, we describe briefly our extensions of the mixed-space MIPS planner, the PDDL 2.1 benchmarks tested, and our experimental results. For comparison, results on applying our approach on the discrete-space ASPEN planner has been reported elsewhere.[6]

MIPS[9] 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.

MIPS can handle the STRIPS subset of PDDL and can cope with numeric quantities and durations in PDDL 2.1. We use MIPS in our experiments because it performs well on PDDL 2.1 benchmarks and its source code is readily available.

5.1. *Implementation details*

Figure 4 shows SGPlang(MIPS), our planner for resolving partitioned subgoals, using MIPS as the basic planner. SGPlang(MIPS) generates an ordered list of goals, decomposes the ℓ_1-penalty formulation of a problem into multiple subproblems, solves each locally, and resolves unsatisfied global constraints by updating their penalties. We have made significant changes to our previous implementation SGPlan$_t$(MIPS)[6] that partitions a planning problem by dividing its temporal horizon into stages and that groups the problem variables based on their temporal bindings. In SGPlan$_t$(MIPS), the only global constraints are those that relate two

```
1.  procedure SGPlang(MIPS)
2.      compute the relevant actions for each goal fact;
3.      compute the partial orders among goal facts;
4.      generate an initial ordered goal list of goal facts;
5.      set iter ⟵ 0;
6.      repeat
7.          for each goal fact in the goal list
8.              call modified MIPS to solve the subproblem;
9.          end_for
10.         if (feasible plan found)
11.             call PERT to generate & evaluate a parallel plan;
12.             decrease some penalties;
13.         else increase penalties γ on unsatisfied
                    global constraints;
14.         iter ← iter + 1;
15.         if (iter % τ == 0) dynamically re-order the goals;
16.     until no change on z and γ in an iteration;
17. end_procedure
```

Fig. 4. SGPlang(MIPS): Our planner for resolving partitioned subgoals using MIPS as the basic planner.

states across stage boundaries. Since MIPS is a heuristic planner that always finds a feasible path up the final state, such a partitioned search often pushes inconsistencies to the last stage, thereby getting the search stuck in an infeasible path that is sometimes difficult to escape. Also, the propagation of information on constraint violations is inefficient because it is done stage-by-stage sequentially.

In our current implementation, we partition the search space based on the goal state instead of the temporal horizon. Specifically, we formulate a subproblem for a goal fact in such a way that there is only one goal state in each stage. We then order the goals into a sequence and find a feasible subplan for each goal fact iteratively.

In each stage, we use local constraints to enforce valid transitions from the initial state to the goal state. We also add global constraints to enforce the solutions of all subproblems to be conflict-free; that is, the solution plan of a subproblem will not invalidate the goal fact of another subproblem.

Note that our approach is different from incremental planning schemes [19] that use a goal agenda. In incremental planning, a set of target facts are maintained, and goal states are added incrementally into the target set. The planner then extends the solution incrementally using an enlarged target set. As a result, once a goal state is satisfied, it will always be satisfied in subsequent extended plans. Such an approach is deficient because the search space is increasingly larger as more goal states are added. Moreover, it is difficult to tell which goals should be satisfied before others.

In contrast, our planner always tries to resolve one goal fact in a stage at a time, while incorporating related global constraints in the objective of the local problem (see (20)). As a result, the search space of subsequent stages is not increasing, and a substantial portion of irrelevant actions in each stage can be eliminated.

Moreover, we add global constraints to relate each pair of goal facts and resolve their inconsistencies in the ℓ_1-penalty formulation and the global search. Violated global constraints are also incorporated during each local search because they act as biases in the objective of each local problem.

The following is a summary of the key techniques studied in this paper.

a) *Search-space reduction for a subproblem* (Steps 2 of Figure 4). Since there is only one goal state for each subproblem, the relevant actions and facts can be reduced substantially beforehand. We perform a *backward relevance analysis* to exclude some irrelevant actions before applying MIPS to solve a subproblem. We maintain an *open list* of unsupported facts, a *close list* of relevant facts, and a *relevance list* of relevant actions. At the beginning, the open list contains a single goal fact, and the relevance list is empty. In each iteration, for each fact in the open list, we find all the actions supporting that fact and not already in the relevance list. We then add these actions to the relevance list, and add the action preconditions that are not in the close list to the open list. We move a fact from the open list to the close list when it is processed. The analysis ends when the open list is empty. At that point, the relevance list will contain all possible relevant actions, while excluding those irrelevant actions. Notice that such a reduction analysis is not tight in the sense that there may still be some irrelevant actions in the relevance list.

The relevance list can be further reduced if we perform a forward analysis to find applicable actions from the initial states before the backward analysis. However, such forward analysis is not helpful because MIPS is a forward heuristic planner.

This analysis takes polynomial time and only needs to be performed once before the search starts. The relevance list for each goal fact is stored and will be used throughout the search.

b) *Ordering of goals.* In order to resolve more difficult goals before easier ones during our search, we define heuristically some partial orders among goal facts (Step 3) and a random order otherwise. Based on the backward relevance analysis, we compute the number of irrelevant actions of each goal fact, and order A before B if A has less irrelevant actions. For goal facts with the same number of irrelevant actions, we apply a second level of partial ordering. Specifically, for A and B with the same number of irrelevant actions, we order A before B if $n_p(A) > n_p(B)$. Here, $n_p(A)$ is the minimum number of preconditions of those supporting actions defined as follows:

$$n_p(A) = \min_{a \in S(A)} n_{pre}(a), \tag{21}$$

where $S(A)$ is the set of all actions that support goal fact A, and n_{pre} is the number of preconditions of action a. The idea is to first resolve more difficult goals, with less irrelevant actions and larger n_p.

At the beginning of a search, we randomly generate a total ordering of the goal facts that satisfy the partial orders (Step 4). We also periodically generate new total orders during the search (Step 15).

c) *Modified MIPS* (Step 8). MIPS carries out a standard A^* heuristic search, where state s is evaluated by heuristic function $H(s)$ based on a relaxed plan extracted from s to the goal state. In SGPlang(MIPS), we use a modified MIPS with two important changes in order to adapt it to our formulation.

First, to guide descents in the ℓ_1-penalty space of each subproblem, we modify the heuristic function for state s as follows:

$$H'(s) = H(s) + D(s) + \sum_{i=1}^{N_G} (\gamma_i a_i + \zeta_i h_i), \tag{22}$$

where $H(s)$ is the original heuristic function of MIPS, $D(s)$ is a heuristic function for penalizing action dependencies, N_G is the number of goals in the original planning problem, γ_i and ζ_i are the penalties for the i^{th} goal fact G_i, a_i is 1 when the action to reach s makes G_i invalid and 0 otherwise, and h_i is 1 when the relaxed plan in MIPS from s to the goal state of s makes G_i invalid and 0 otherwise.

Second, in expanding a node in MIPS, we refer to the relevance list generated before and prune all actions not in the relevance list of the goal fact.

d) *Heuristic objective.* We include a heuristic objective $D(s)$ in (22) to measure solution quality:

$$D(s) = \alpha_D * n_d, \tag{23}$$

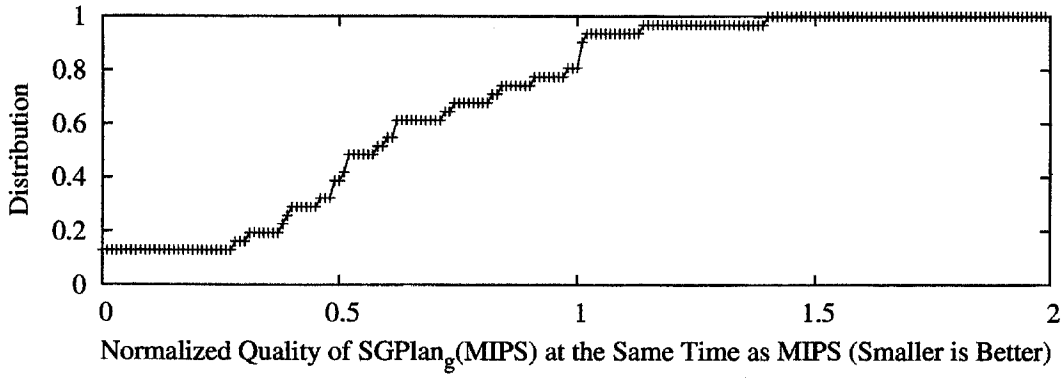
where α_D is a weighting factor (0.01 in our experiments), and n_d is the number of actions in the relaxed plan of s that are dependent on actions in other subplans. The idea here is to favor solution plans with less dependencies because independent actions can be scheduled in parallel by PERT, leading to solution plans with shorter durations and higher quality. In the future, we plan to study better objective functions. One possibility is to apply PERT and compute the objective function at each s and define $D(s)$ to be the resulting quality.

e) *Penalty updates.* For goal fact i, we assign penalties γ_i and ζ_i as in (22). When a feasible plan is not found in an iteration (Steps 7-9), we increase (but may periodically decrease) the penalties for those unsatisfied goal facts (Step 13). Further, when a feasible plan has been found, we reduce some of the penalties, randomly select one goal fact, and reset its penalties to zero. This allows the search to move quickly from one local minimum to another (Step 12).

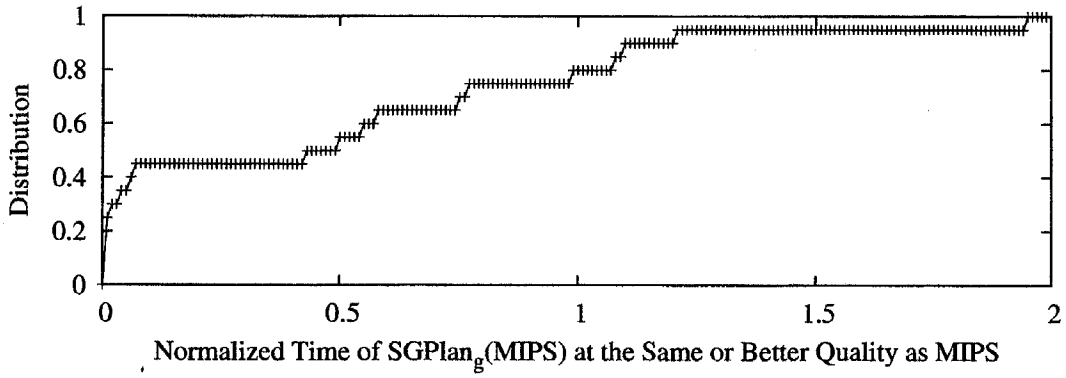
5.2. *Experimental results*

We show that SGPlang(MIPS) improves significantly over the original MIPS on a set of PDDL2.1 benchmarks used in the Third International Planning Competition. The problems studied include *DriveLogNumeric, DriveLogSim, DriveLogTime, ZenoTravelNumeric, ZenoTravelSim*, and *ZenoTravelTime.*

We have used the most recent executable of MIPS downloaded from its Web site and ran it with default parameters and a maximum time limit of 10^7ms. All experiments were done on an AMD Athlon MP2000 PC with Linux Redhat 7.2.



a) Distribution of the quality of solutions found by SGPlang(MIPS), normalized with respect to those of MIPS. Each problem evaluated by SGPlang(MIPS) was limited by the same amount of time taken by MIPS for that problem.



b) Distribution of the normalized times taken by SGPlang(MIPS) to find solutions of the same or better quality as those found by MIPS. SGPlang(MIPS) was allowed to evaluate a problem until a solution with the same or better quality with respect to that of MIPS had been found.

Fig. 5. Normalized times and qualities of SGPlang(MIPS) with respect to MIPS on the 33 problems solvable by MIPS in more than 1 sec. but less than 10^3 sec. The times and qualities of MIPS are normalized to one in both plots.

For the 114 problems studied (see Table 1), we divide them into three sets: a) 63 solvable by MIPS in 1 second; b) 33 solvable by MIPS in 10^3 seconds; and c) 18 unsolvable by MIPS in 10^3 seconds. For problems in class (a), SGPlang(MIPS) usually takes longer time due to its overhead but can find better-quality plans in 60 problems. For those in class (b), Figure 5 plots the distribution of normalized quality (*resp.* normalized time) of solutions found by SGPlang(MIPS). The results show that SGPlang(MIPS) is able to improve over MIPS in 80.5% of the cases in quality or 80.1% in time. For problems in class (c), SGPlang(MIPS) can solve 11 of them in 10^3 seconds. There is no problem solvable by MIPS but not by SGPlang(MIPS).

Table 1: Results on MIPS and SGPlang(MIPS) in solving some PDDL2.1 benchmark problems. All timing results are in milliseconds. Both solvers were ran with a maximum time limits of 10^6ms. "-" means that no solution was found at the time limit. For MIPS, *Time* and *Sol* list the solution time and quality (lower is better). For SGPlang(MIPS), $Time_1$ 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 final solution found within the time limit. For each problem, a boxed number indicates the better quality between MIPS and SGPlang(MIPS).

Problem ID	MIPS		SGPlang(MIPS)			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f
DriveLogTime1	65	303	120	303	9360	302
DriveLogTime2	80	310	120	330	41200	253
DriveLogTime3	75	173	130	207	430	173
DriveLogTime4	75	392	130	332	7340	230
DriveLogTime5	103	112	130	342	15180	102
DriveLogTime6	124	260	130	239	121390	168
DriveLogTime7	123	268	130	307	91890	208
DriveLogTime8	235	313	150	351	9560	201
DriveLogTime9	233	980	230	725	13550	334
DriveLogTime10	287	340	250	452	16760	98
DriveLogTime11	343	391	240	500	37530	232
DriveLogTime12	1530	611	340	2173	18220	290
DriveLogTime13	1256	558	420	658	46350	367
DriveLogTime14	2303	1049	1260	1113	92420	284
DriveLogTime15	9853	893	1010	703	108550	236
DriveLogTime16	-	-	-	-	-	-
DriveLogTime17	236244	954.94	3720	2425	264040	943
DriveLogTime18	-	-	33440	1809	400050	947
DriveLogTime19	-	-	-	-	-	-
DriveLogTime20	-	-	95490	1745	465300	1375
DriveLogSim1	90	92.07	120	93	6540	91
DriveLogSim2	90	92.21	120	104	230	93
DriveLogSim3	98	40.07	120	48	340	40
DriveLogSim4	99	89.16	130	97	2120	52
DriveLogSim5	112	51.19	130	113	10430	51
DriveLogSim6	117	64.13	130	90	26800	52
DriveLogSim7	122	40.09	130	50	170	40
DriveLogSim8	279.1	111.26	240	127	226260	52
DriveLogSim9	202	264.31	240	185	85870	92
DriveLogSim10	269.1	61.21	230	49	450	39
DriveLogSim11	351	99.21	240	82	397190	65
DriveLogSim12	1772	252.41	370	564	319680	175
DriveLogSim13	1734	104.29	330	269	53760	102
DriveLogSim14	2403	226.44	2050	1587	289930	95
DriveLogSim15	13620	265.43	690	319	228280	115
DriveLogSim16	-	-	-	-	-	-
DriveLogSim17	549119	223.94	4110	875	260770	241
DriveLogSim18	-	-	66930	702	214540	327

continued on next page

continued from previous page

Problem ID	MIPS		SGPlang(MIPS)			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f
DriveLogSim19	-	-	54560	1006	61680	1068
DriveLogSim20	-	-	173190	771	274660	280
DriveLogNumeric1	90	1099	120	953.07	132170	771.2
DriveLogNumeric2	89	1497	120	2025.57	22250	999.69
DriveLogNumeric3	92	907	110	1234.75	5580	629.93
DriveLogNumeric4	112	715	130	1055.4	4930	705
DriveLogNumeric5	124	878	130	1330.12	258770	581.2
DriveLogNumeric6	130.1	1667	220	976.76	150890	966.48
DriveLogNumeric7	123.1	866	220	100938.82	138940	895.31
DriveLogNumeric8	19680	3273	230	2242.08	1510	1430.2
DriveLogNumeric9	629	3002	240	3164.72	37660	1772.58
DriveLogNumeric10	278	402	250	334.16	12190	139.05
DriveLogNumeric11	7250	616	260	562.58	137320	313.33
DriveLogNumeric12	14320	3227	240	5015.88	7810	2050.31
DriveLogNumeric13	2521	2148	380	2023.86	87080	1111.22
DriveLogNumeric14	34433.1	3347	1210	11115.07	57000	1696.94
DriveLogNumeric15	12421	1753	600	1593.36	298880	1126.57
DriveLogNumeric16	-	-	-	-	-	-
DriveLogNumeric17	-	-	20580	19689.35	631190	7703.86
DriveLogNumeric18	-	-	10450	12836.92	264460	8830.81
DriveLogNumeric19	-	-	64680	26282.09	471550	25177.89
DriveLogNumeric20	-	-	207710	18601.27	504830	14943.57
ZenoTravelTime1	50	27.257	110	28.14	130	27.256
ZenoTravelTime2	50	30.2104	120	31.7	140	30.5
ZenoTravelTime3	78	18.1527	130	32.52	14380	17.62
ZenoTravelTime4	82	153.294	130	223.08	14210	73.03
ZenoTravelTime5	99	37.7473	140	22.74	230	18.31
ZenoTravelTime6	93	51.7826	120	66.65	118150	41.32
ZenoTravelTime7	112	142.179	240	115.76	48800	86.12
ZenoTravelTime8	201	160.639	210	243.91	457680	137.67
ZenoTravelTime9	223	119.82	280	109.14	221920	55.89
ZenoTravelTime10	221	181.68	240	246.85	76090	131.14
ZenoTravelTime11	276	155.308	210	173.99	153040	115.82
ZenoTravelTime12	353	126.007	340	209.92	195060	79.06
ZenoTravelTime13	455	90.28	210	134.16	244140	56.99
ZenoTravelTime14	7823	375.056	1920	754.95	358380	360.51
ZenoTravelSim1	80	180.01	110	173.32	130	173
ZenoTravelSim2	78	643.06	210	1019	68920	596
ZenoTravelSim3	1431	683.09	120	1048	127740	280
ZenoTravelSim4	124	936.11	120	1333	76590	566
ZenoTravelSim5	234	690.13	180	2664	10580	399
ZenoTravelSim6	330.1	480.12	230	849	15450	326
ZenoTravelSim7	213	716.16	180	1557	16380	629

continued on next page

continued from previous page

Problem ID	MIPS		SGPlang(MIPS)			
	Time	*Sol*	$Time_1$	Sol_1	$Time_f$	Sol_f
ZenoTravelSim8	1243	846.13	130	923	405430	570
ZenoTravelSim9	1376	1256.24	260	1965	8960	568
ZenoTravelSim10	1523	1432.29	250	2163	299780	659
ZenoTravelSim11	3734.1	1219.19	240	1741	67670	446
ZenoTravelSim12	3551	1179.29	370	2955	146260	615
ZenoTravelSim13	3603	913.31	300	2565	122150	658
ZenoTravelSim14	124518.4	1099.36	6060	1758	33210	910
ZenoTravelSim15	233530	1758.4	31000	1921	99690	984
ZenoTravelSim16	-	-	22400	2132	251730	1427
ZenoTravelSim17	-	-	791450	5388	821840	4418
ZenoTravelSim18	-	-	-	-	-	-
ZenoTravelSim19	-	-	-	-	-	-
ZenoTravelSim20	-	-	-	-	-	-
ZenoTravelNumeric1	72	13564	100	14101.66	120	13563.9
ZenoTravelNumeric2	70	6786	150	17363.04	4320	6881.55
ZenoTravelNumeric3	92	7505	130	11049.28	310990	4505.43
ZenoTravelNumeric4	91	16964	120	19063.51	122370	16960.57
ZenoTravelNumeric5	100	19916	120	12957.98	227850	3693.79
ZenoTravelNumeric6	112	35282	130	126461.99	105150	14748.29
ZenoTravelNumeric7	103.1	16472	130	14950.24	393460	7714.38
ZenoTravelNumeric8	183	33543	140	52760.33	494050	18871.38
ZenoTravelNumeric9	192	28047	170	20192.73	363850	4552.97
ZenoTravelNumeric10	214.1	79564	170	95806.62	521700	36412.08
ZenoTravelNumeric11	252	55480	350	135540.76	173460	16786.29
ZenoTravelNumeric12	306	41310	400	63680.37	150250	20810.31
ZenoTravelNumeric13	413	82230	340	136451.66	371210	18426.04
ZenoTravelNumeric14	6247	233381	1110	234779.58	175790	148784.93
ZenoTravelNumeric15	15890	147618	3100	152545.52	375150	66533.79
ZenoTravelNumeric16	31652	143282	13990	134016.82	246200	74784.77
ZenoTravelNumeric17	64438	182558	298350	221784.16	625040	187502.56
ZenoTravelNumeric18	123543.4	70794	58610	164832.79	447350	155460.01
ZenoTravelNumeric19	135935	212997	89970	324208.44	544790	197613.24
ZenoTravelNumeric20	245335	89937	235460	689010.03	945640	526116.64

6. Conclusions

In this paper, we have presented the theory of extended saddle points in mixed space. By defining a mixed neighborhood in partitioned variable space, we show a set of necessary conditions, one for each partition, that collectively are sufficient.

The theory leads to an efficient iterative scheme for resolving global constraints across subproblems partitioned by constraints and for finding extended saddle points in each partitioned subproblem. Using the mixed-space MIPS planner to solve partitioned planning problems, we have demonstrated significant improve-

ments on some PDDL2.1 benchmark problems, both in terms of the quality of the plans generated and the execution times to find these plans.

The partitioning approach presented is important for reducing the exponential complexity of nonlinear constrained optimization problems. By partitioning a problem into subproblems and by reducing the search space of each partitioned subproblem using our proposed theory, we can reduce the complexity of the overall problem. Further, since constraint partitioning leads to planning subproblems of similar nature but of smaller scale, we can exploit existing planners and their efficient pruning techniques to further reduce the search space of these subproblems.

References

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* J. Wiley and Sons, 1989.
2. M. Avriel. *Nonlinear Programming: Analysis and Methods.* Prentice-Hall, Inc., Englewood Cliffs, NJ., 1976.
3. D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Massachusetts, 1999.
4. A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
5. B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence, Special issue on Heuristic Search*, 129(1), 2001.
6. Y. X. Chen and B. W. Wah. Automated planning and scheduling using calculus of variations in discrete space. In *Proc. Int'l Conf. on Automated Planning and Scheduling*, pages 2–11, June 2003.
7. S. Chien, *et al.* ASPEN - Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps.* Toulouse, France, 2000.
8. P. Doherty and J. Kvarnstrm. Talplanner: An empirical investigation of a temporal logic-based forward chaining planner. *Proc. Sixth Int'l Workshop on Temopral Logic-based Forword Chaining Planner*, pages 47–54, 1999.
9. S. Edelkamp. Mixed propositional and numerical planning in the model checking integrated planning system. *Proc. AIPS Workshop on Planning for Temporal Domains*, 2002.
10. M. P. Fourman. Propositional planning. *Proc. Workshop on Model Theoretic Approaches to Planning, AIPS 2000*, 2000.
11. M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Tech. Rep., Dept. of Computer Science, Univ. of Durham, Durhan, UK*, February 2002.
12. A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs with action costs. *Proc. of the Sixth Int. Conf. on AI Planning and Scheduling*, pages 12–22, 2002.
13. N. I. M. Gould, D. Orban, and 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.
14. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.
15. A. K. Jonsson, P. H. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. 2nd Int'l NASA Workshop on Planning and Scheduling for Space.* NASA, 2000.

16. H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, ar stochastic search. *Proceedings of the 13th National Conference on Artificial Intell gence (AAAI-96)*, pages 1194–1201, 1996.
17. H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Pro Int'l Joint Conf. on Artificial Intelligence*. IJCAI, 1999.
18. H. Kautz and J. P. Walser. Integer optimization models of AI planning problems. *Th Knowledge Engineering Review*, 15(1):101–117, 2000.
19. J. Koehler and J. Hoffmann. On reasonable and forced goal ordering and their use i an agenda-driven planning algorithm. *J. of AI Research*, 12:339–386, 2000.
20. F. Lin. A planner called R. *AI Magazine*, pages 73–76, 2001.
21. D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *J. of A Research (JAIR)*, 1998.
22. D. Nau, H. Muoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning wit partially ordered subtasks. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, page 425–430. IJCAI, 2001.
23. R. S. Nigenda, X. Nguyen, and S. Kambhampati. AltAlt: Combining the advantage of Graphplan and heuristic state search. Technical report, Arizona State University 2000.
24. J. Penberethy and D. Weld. UCPOP: A sound, complete, partial order planner fo ADL. In *Proc. 3rd Int. Conf. on Principle of Knowledge Representation and Reason ing*, pages 103–114, 1992.
25. J. Penberethy and D. Weld. Temporal planning with continuous change. In *Proc. 12th National Conf. on AI*, pages 1010–1015. AAAI, 1994.
26. R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.
27. I. Refanidis and I. Vlahavas. The GRT planner. *AI Magazine*, pages 63–66, 2001.
28. D. Smith, J. Frank, , and A. Jonsson. Bridging the Gap between Planning and Scheduling. *Konwledge Engineering Review*, 15(1):47–83, 2000.
29. M. B. D. Subbarao and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. Technical report, Arizona State University, 2002.
30. A. Tate, B. Drabble, and R. Kirby. O-Plan2: an open architecture for command, planning and control. *Intelligent Scheduling*, pages 213–239, 1994.
31. B. W. Wah and Y. X. Chen. Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *Proc. IEEE Int'l Conf. on Tools with Artificial Intelligence*, pages 266–273, November 2003.
32. B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, pages 28–42, October 1999.
33. D. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, pages 232–246, 1990.
34. S. Wolfman and D. Weld. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, 15(1), 2000.