SOLVING NONLINEAR CONSTRAINED OPTIMIZATION PROBLEMS THROUGH CONSTRAINT PARTITIONING

BY

YIXIN CHEN

B.S., University of Science and Technology of China, 1999
M.S., University of Illinois at Urbana-Champaign, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

In this dissertation, we propose a general approach that can significantly reduce the complexity in solving discrete, continuous, and mixed constrained nonlinear optimization (NLP) problems. A key observation we have made is that most application-based NLPs have structured arrangements of constraints. For example, constraints in AI planning are often localized into coherent groups based on their corresponding subgoals. In engineering design problems, such as the design of a power plant, most constraints exhibit a spatial structure based on the layout of the physical components. In optimal control applications, constraints are localized by stages or time.

We have developed techniques to exploit these constraint structures by partitioning the constraints into subproblems related by global constraints. Constraint partitioning leads to much relaxed subproblems that are significantly easier to solve. However, there exist global constraints relating multiple subproblems that must be resolved. Previous methods cannot exploit such structures using constraint partitioning because they cannot resolve inconsistent global constraints efficiently.

We have developed a mathematical theory that provides strong necessary and sufficient analytical conditions for limiting the subspace to be searched when resolving the global constraints. We have proposed the theory of extended saddle points (ESP) to support constraint partitioning when solving NLPs. Based on a novel penalty formulation, ESP offers a necessary and sufficient condition for constrained local optima of NLPs in discrete,

continuous, and mixed spaces. It facilitates constraint partitioning by providing a set of necessary conditions, one for each subproblem, to characterize the local optima. It further reduces the complexity by defining a much smaller search space in each subproblem for backtracking. Since resolving the global constraints only incurs a small amount of overhead, our approach leads to a significant reduction of complexity.

Our partition-and-resolve approach has achieved substantial improvements over existing methods in solving AI planning and mathematical programming problems. In this dissertation, we present SGPlan, a planner based on constraint partitioning that has significantly improved the solution time and quality on many application domains when compared to other leading planners. We also describe our implementation that has successfully incorporated ESP with ASPEN, a planning system for spacecraft exploration developed by NASA. The ESP planner performs 100 to 1000 times faster than the original ASPEN on NASA's benchmarks and can generate solutions of much higher quality.

Constraint partitioning has led to a major breakthrough in solving mathematical programming problems in operations research and engineering applications. In this dissertation, we have applied our method to solve some large-scale continuous and mixed-integer NLPs in standard benchmarks. We have solved some large-scale problems that were not solvable by other leading optimization packages and have improved the solution quality on many problems.

To my wife, Juan, and my parents

Acknowledgments

I would like to thank my advisor, Professor Benjamin W. Wah, for his guidance during the course of my graduate study. He taught me how to find important subjects, define suitable research problems, solve problems, and present the results in an informative and interesting way. The problem solving skills I learned from him will be invaluable for my career as a scientist. He also taught me to be confident, persistent, and hard-working in performing research.

I would like to thank Professors Jiawei Han, Grigore Rosu, and Martin Wong for serving on my Ph.D. committee and for providing many useful comments and suggestions. I would like to thank Professor C. V. Ramamoorthy for his encouragement and support during my graduate study.

I would also like to thank Dr. Tao Wang, Dr. Xiao Su, Dr. Zhe Wu, Dr. Dong Lin, Dr. Minglun Qian, Dr. Honghai Zhang, Hang Yu, Dong Xin, Chih-Wei Hsu, Batu Sat, Mark Richard, and all other members in our research group for providing insightful comments on the work and for providing a friendly environment for me to work in.

I wish to thank my wife, Juan, and my parents for their love and support.

Finally, I would like to acknowledge the support of the National Science Foundation Grant IIS 03-12084 and National Aeronautics and Space Administration Grant NCC 2-481.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Formulation

Nonlinear optimization is an important problem that has abundant applications in science and engineering. In this thesis, we study general mixed-integer nonlinear programming problems (MINLPs) of the following form:

$$(P_m): \quad \min_{z} \quad f(z), \tag{1.1}$$
$$\text{subject to} \quad h(z) = 0 \ \text{ and } \ g(z) \leq 0,$$

where variable $z = (x, y)$, $x \in \mathcal{R}^v$ is the continuous part, and $y \in \mathcal{D}^w$ is the discrete part. The objective function f is lower bounded and is continuous and differentiable with respect to x, whereas the constraint functions $g = (g_1, \ldots, g_r)^T$ and $h = (h_1, \ldots, h_m)^T$ are general functions that can be discontinuous, non-differentiable, or not even in closed form.

MINLPs defined in (1.1) cover a large class of nonlinear optimization problems, with discrete nonlinear programming problems (DNLPs) and continuous nonlinear programming problems (CNLPs) as special cases. Ample applications exist in operations research, planning and scheduling, optimal control, engineering designs, and production management. The applications we have studied in this research include AI and NASA-related planning and scheduling problems, and engineering design applications formulated as NLPs.

The AI planning problems are modelled in some standard modelling languages, such

as PDDL, and are usually drawn from real-world applications, such as airport operations, petroleum delivery, optical-network routing, satellite operations, electricity networks, mobile communication networks, logistics, and games. An AI planning problem typically involves finding a sequential or parallel schedule of actions to achieve a set of subgoals, subject to logical, temporal, and numerical constraints among actions.

The NASA-related planning problems of spacecraft operations entail finding a sequence of low-level actions to achieve some user-defined high-level scientific goals subject to temporal constraints among actions, while optimizing a objective function at the mean time. At NASA, some of the primary systems that require autonomous operations include deep-space probes, planetary rovers, and deep-space communication antennae. Automated planning/scheduling technologies have great promise in reducing operations cost while increasing the autonomy of aerospace systems. Automating the sequence-generation process allows spacecraft commands by non-operations personnel, hence allowing significant reductions in mission operations workforce.

Many engineering design problems are also formulated and solved as NLPs. Example applications include trajectory optimization for robots and missiles, structural optimization, power flow design, financial portfolio management, principle component analysis, filter design, facility location selection, and optimal control. The constraint and objective functions can be linear, quadratic, and nonlinear, the variables can be discrete, continuous, or mixed, and the typical size of a problem ranges from 10 to 5000 variables and constraints.

MINLPs defined in (1.1) are difficult and expensive to solve. Their prohibitive search complexity is a key factor that hinders the development of many important computer science and engineering applications. For instance, automated planning and theorem proving in AI are not practical for large applications due to their high computational costs involved in solving problems that can be formulated as discrete NLPs. As another example, our

capabilities in nonlinear optimal control are very limited because we do not have efficient ways for solving multistage nonlinear optimization problems.

MINLPs are technologically difficult to solve for several reasons. First, they involve huge search spaces that grow exponentially with respect to the number of variables. Existing solvers have difficulties in solving large instances because of the exponential complexity. Second, the constraints in (1.1) may not be continuous and differentiable, which can be utilized to facilitate an efficient search. These constraints could be discontinuous, non-differentiable, and not even in closed form. Third, we do not assume any special properties on these functions, such as linearity and convexity. The functions can be nonlinear or symbolic.

In this research, we have made a key observation that most MINLPs from real-world applications have strong structures and locality of constraints. Motivated by this observation, we propose a general approach to significantly reduce the search complexity by exploiting the constraint structure and by partitioning the problem constraints.

1.2 Observations on Constraint Structure

We motivate this research work by examining the constraint structure of two example problems and by showing how the partitioning of constraints can lead to a significant reduction of solution time.

1.2.1 Example I: The TRIMLON Benchmark

We start by showing the problem structure of an example MINLP called TRIMLON12 that cannot be solved by existing solvers. This is an instance of the TRIMLON benchmark [36] with $I = 12$ and $J = 12$. The goal is to produce a set of product paper rolls from raw paper

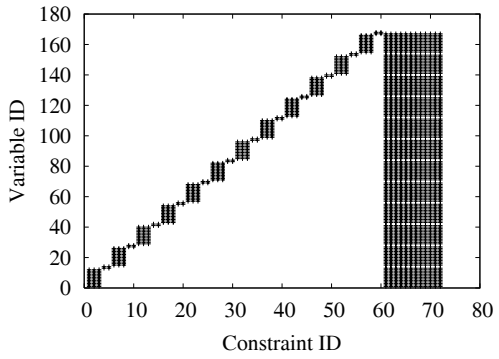


Figure 1.1: Regular structure of constraints in TRIMLON12. A dot in the graph represents a variable associated with a constraint.

rolls by assigning continuous variables m and y and integer variables n in order to minimize f as a function of the trim loss and the overall production cost.

$$
\begin{aligned}
&\text{variables:} && y[j],\ m[j],\ n[j,i] \ \text{ where } \ i = 1, \cdots, I;\ j = 1, \cdots, J && \\
&\text{objective:} && \min_{z=(y,m,n)} f(z) = \textstyle\sum_{j=1}^{J}(c[j] \cdot m[j] + C[j] \cdot y[j]) && \text{(OBJ)} \\
&\text{subject to:} && B_{min} \le \textstyle\sum_{i=1}^{I}(b[i] \cdot n[i,j]) \le B_{max} && \text{(C1)} \\
& && \textstyle\sum_{i=1}^{I} n[i,j] - N_{max} \le 0 && \text{(C2)} \\
& && y[i] - m[j] \le 0 && \text{(C3)} \\
& && m[j] - M \cdot y[j] \le 0 && \text{(C4)} \\
& && Nord[i] - \textstyle\sum_{j=1}^{J}(m[j] \cdot n[i,j]) \le 0. && \text{(C5)}
\end{aligned}
$$

An instance of TRIMLON can be specified by defining I and J, leading to $(I + 2)J$ variables and $5J + I$ constraints. For example, there are 168 variables and 72 constraints in TRIMLON12.

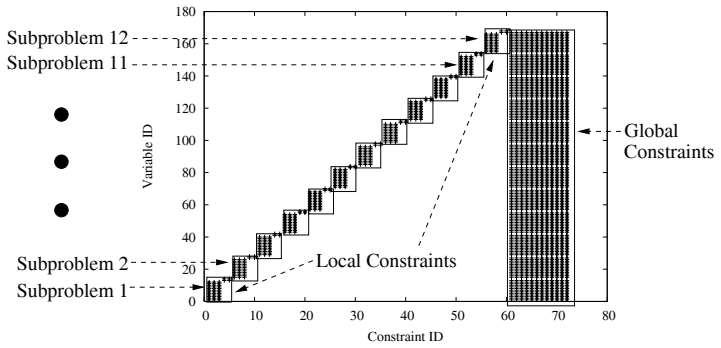


Figure 1.2: An illustration of the partitioning of the constraints in TRIMLON12 into 12 subproblems.

A key observation we have made on many application benchmarks, including TRIMLON12, is that their constraints do not involve variables that are picked randomly from their variable sets. Invariably, many constraints in existing benchmarks are highly structured because they model spatial and temporal relationships that have strong locality, such as those in physical structures and task scheduling.

Figure 1.1 illustrates this point by depicting the constraint structure of TRIMLON12. It shows a dot where a constraint (with unique ID on the x axis) is related to a variable (with a unique ID on the y axis). With the order of the variables and the constraints arranged properly, the figure shows a strong regular structure of the constraints.

Based on the regular constraint structure of a problem instance, we can cluster its constraints into multiple loosely coupled groups. To illustrate the idea, consider the partitioning of the constraints in TRIMLON12 by index $j \in S_J = \{1, \cdots, 12\}$. Suppose S_J is partitioned into N disjoint subsets such that $S_1 \cup \cdots \cup S_N = S_J$. Then the k^{th} subproblem, $k = 1, \ldots, N$, has variables $y[j], m[j], n[j, i]$, where $i = 1, \cdots, I,\ j \in S_k$, and the same objective function

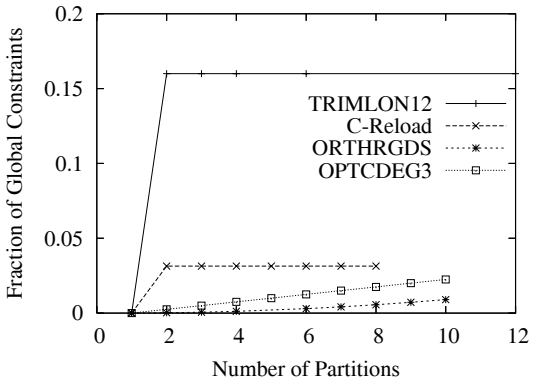
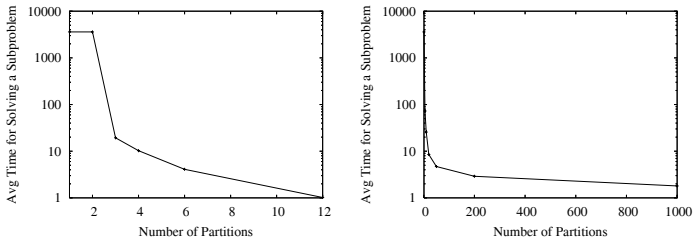


Figure 1.3: Monotonic increase in the fraction of constraints that are global with respect to increasing number of partitions on four benchmarks.

(OBJ). (C1)-(C4) are its local constraints because each involves only local indexes on j. (C5), however, is a global constraint because it involves a summation over all j.

Figure 1.2 illustrates the partitioning of TRIMLON12 into $N = 12$ stages, where S_J is partitioned evenly and $S_k = \{k\}$. Out of the 72 constraints, 60 are local constraints and 12 are global constraints. Hence, the fraction of constraints that are global is 16.7%.

The fraction of global constraints in a problem instance depends strongly on its constraint structure and the number of partitions. Using the straightforward scheme as in TRIMLON12 to partition the constraints evenly, Figure 1.3 illustrates the fraction of global constraints either increases monotonically or stays unchanged with respect to the number of partitions for the four benchmarks [55], including TRIMLON, C-Reload (a MINLP problem modelling nuclear core reload pattern optimization), ORTHRGDS (a CNLP problem of finding the least square error in orthogonal regression), and OPTCDEG3 (a CNLP problem for nonlinear optimal control in a noise damping system).



a) TRIMLON12(MINLP) b) ORTHRGDS(CNLP)

Figure 1.4: Exponential decrease in the average time to solve a subproblem with respect to increasing number of partitions on the TRIMLON12 and the ORTHRGDS benchmarks.

In contrast, the time required to solve a subproblem decreases monotonically as the number of partitions is increased. When a problem is partitioned by its constraints, each subproblem is much more relaxed than the original problem and can be solved in exponentially less time than the original. Figure 1.4 illustrates this exponential decrease on the average time for solving a subproblem with increasing number of partitions. The overheads between no partitioning and partitioning can be several orders of magnitude.

1.2.2 Example II: The AIRPORT Planning Problem

Our second example is an AI planning problem called AIRPORT from the test problem suite used by the Fourth International Planning Competition (IPC4) [23]. The AIRPORT problem models an airport scheduling domain, where there are multiple airplanes to be moved around in an airport.

Figure 1.5 illustrates the topology of an airport in the AIRPORT4 instance. The example involves a planning task for moving three airplanes from their starting positions to some destination gates. To apply a planning system to solve the problem, we first model this

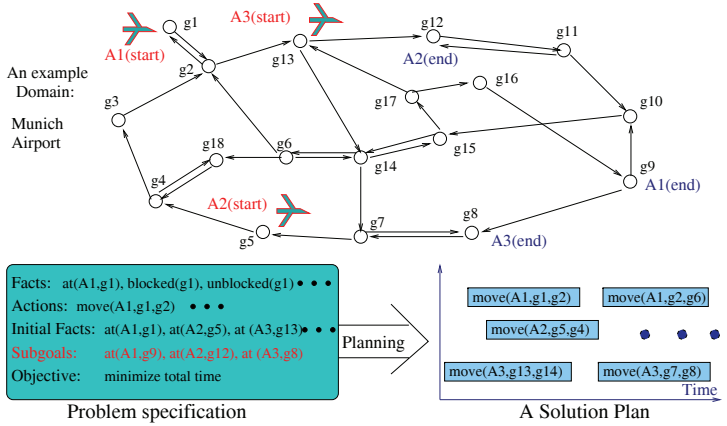
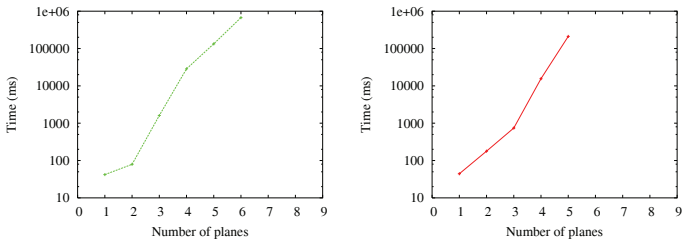


Figure 1.5: The topology of the airport in the AIRPORT4 planning problem instance.

problem in a modelling language to specify the facts, actions, initial state, and goals of the planning problem. Note that the goal includes multiple subgoals, one for each airplane. A solution plan, shown in the right bottom part of Figure 1.5, is a temporal plan where actions have durations and can have overlapping execution times.

Figure 1.6 illustrate the solution time of two existing planners in solving the AIRPORT problem with various number of airplanes. The two solvers are LPG [30], a stochastic planner that performs random probing in the solution space, and FF [37], a heuristic search planner that performs hill climbing based on a relaxed-plan heuristic function. We can see that the time to solve the problem increases exponentially with increasing number of airplanes, which makes the scheduling of multiple airplanes very expensive.

To explain the exponential increase of search complexity of existing planners, we need to closely examine the constraints involved with the problem. In temporal planning problems, the constraints are mutual exclusion constraints. Two actions are mutually exclusive if they



a) LPG (stochastic search) b) FF (heuristic search)

Figure 1.6: The exponential growth in complexity of two existing planners on the AIRPORT domain. The x-axis shows the number of airplanes (subgoals) in the problem instance, and the y-axis shows the solution time to find a feasible solution. We have used LPG1.2 [30] and FF2.0 [37] on an AMD Athlon MP2800 PC running Linux AS4.

interfere with each other and cannot be scheduled in certain ways. Mutual exclusion will be defined in detail in Chapter 4. Here, it suffices to know that a plan is feasible when there are no violated mutual exclusion constraints.

Figure 1.7 plots the mutual exclusion constraints in three AIRPORT planning problems, with one, two, and three subgoals, respectively. We see that the number of actions and the number of constraints grow in proportion to the number of subgoals, which lead to an exponential growth in search complexity by existing planners that solve the planning problem as a whole.

Figure 1.8 shows that the seemingly random constraints in the solution plans are in fact highly structured and can be clustered by their subgoals. To see this, we generate a plan using LPG [30] for each of the three subgoals in the AIRPORT-4 instance, compose the plans together, and plot all the actions and the constraints. We show that most of the constraints are local constraints relating two actions from the same subproblem, and that only

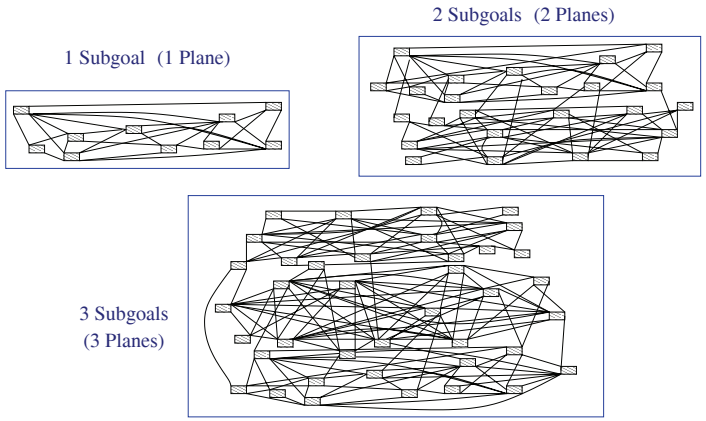


Figure 1.7: Illustration of the mutual exclusion constraints in the AIRPORT planning problem. Each box represents an action, and there is a line between two actions if there is a mutual-exclusion constraint between them.

a few global constraints relate two actions from different subproblems. This observation is intuitively sound because the movements of airplanes are largely independent. Two airplanes interact with each other only when they are at the same position and can be scheduled independently most of the time.

Based on the observation of constraint locality shown in Figure 1.8, we propose in this thesis to partition the constraints by their subgoals into multiple subproblems, one for each subgoal, and solve each subproblem individually before composing the solution plan. Figure 1.9 compares the search time of our planner SGPlan based on the constraint-partitioning approach and those of LPG and FF. We see that the proposed approach leads to a significant reduction of search time and much better scalability.

The performance improvement is due to the fact that, after we partition the problem into

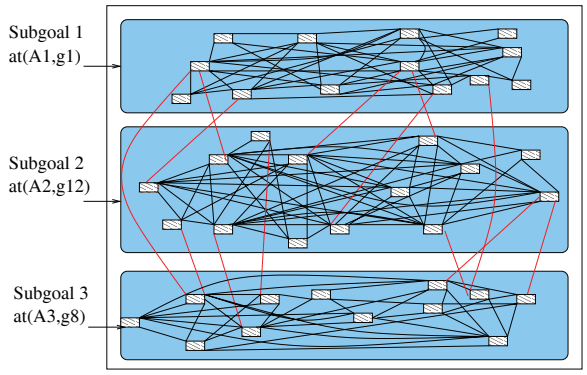


Figure 1.8: Illustration of constraint locality in the AIRPORT4 instance.

multiple subproblems, each subproblem is much more relaxed than the original problem, and can be solved much faster. The subproblems are much more relaxed because they involve a much smaller number of constraints than that of the original problem. Thus, the total time to solve all the subproblems is significantly reduced. However, since there are global constraints that may be violated when multiple subplans are combined, the efficiency in resolving inconsistent global constraints is a key factor for the overall performance of the proposed constraint partitioning approach.

1.3 Research Goal and Problems

The general goal of this thesis is to solve large-scale NLPs from planning and engineering applications efficiently by exploiting the problem structure and the partitioning of the constraints. The metrics to measure the success in this research are the solution time and the solution quality of solving large problems.

The keys to the success of using constraint partitioning to solve MINLPs depend on the

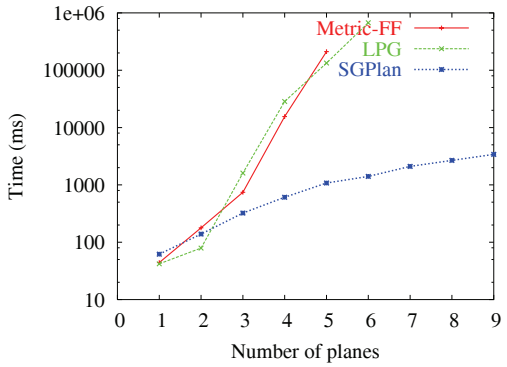


Figure 1.9: Comparison of solution times of LPG, FF, and SGPlan on the AIRPORT planning domain with an increasing number of airplanes.

identification of the constraint structure of a problem instance, the strategy to optimally partition the problem, and the efficient resolution of its violated global constraints. To this end, we study the following problems in this thesis:

a) *Automated partitioning strategy.* There are two components of a partitioning scheme: the dimension in which constraints can be partitioned and the number of partitions. To determine the dimension for partitioning constraints, we study methods for determining the structure of a problem instance after possibly reorganizing its variables and constraints, and identifying the dimension by which the constraints can be partitioned. To determine the optimal number of partitions, we exploit trade-offs among the number of partitions (Figures 1.3), the overhead for evaluating each subproblem (Figure 1.4), and the overhead for resolving the violated global constraints. We study methods for automatically selecting the optimal number of partitions in order to minimize the total time complexity.

b) *Resolution of violated global constraints.* Although constraint partitioning leads to

much simpler subproblems, the approach is not exploited in existing optimization algorithms except in some special cases. The approach leads to global constraints across subproblems that need to be resolved. These global constraints may be either constraints in the original problem that involve variables in different subproblems, or new constraints created to enforce the consistency of internal states across subproblems. The general difficulty lies in the exponentially large space across the subproblems to be searched in resolving violated global constraints. In this research, we study how to efficiently resolve the violated global constraints by developing a mathematical foundation to characterize the local optimal solution under constraint partitioning, and by reducing the subspace to be searched in resolving violated global constraints.

c) *Applications on planning and nonlinear programming.* In this research, we apply the constraint partitioning approach to solve planning problems and large-scale nonlinear programming benchmarks. Planning is a core problem for artificial intelligence, and has wide applications in logistics, mobile communication, transportation, operations management, and aerospace engineering. We explore the temporal and logical locality of constraints in planning problems and study the implementation of fully automated planning systems based on constraint partitioning. We also apply the constraint partitioning approach to solve large-scale CNLP and MINLP benchmarks from engineering applications. In these applications, we study the automated recognition of constraint structures, the determination of optimal partitioning, and the efficient resolution of global constraints.

1.4 Contributions and Significance of Research

The main contributions of this thesis are as follows:

a) We propose an extended saddle-point condition (ESPC) for resolving violated global

constraints efficiently. In this theory, we develop a necessary and sufficient condition that governs all constrained local minima, when a MINLP problem is formulated in a penalty function. ESPC in mixed space extends the previous condition developed for discrete optimization [98, 91, 92, 97, 99, 89, 87, 14, 85, 86, 84, 90, 88]. This research shows that each constrained local minimum of a MINLP problem is associated with a saddle point of a new penalty function and when penalties are sufficiently large. Using this result, one way to look for a constrained local minimum of a MINLP is to increase gradually the penalties of violated constraints in the corresponding unconstrained penalty function and to find repeatedly local minima by an existing algorithm until a feasible solution to the constrained model is found.

b) We show that 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 that is 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. We show that each subproblem is similar to the original problem but of a smaller scale. We further show that penalties always exist under similar relaxed conditions for constrained local minima in subproblems, and that each decomposed ESPC is necessary individually and sufficient collectively. Since the decomposed ESPC is satisfied by constrained local minima in each subproblem (that also satisfy the local constraints), it is much more effective than the local constraints alone for limiting the search space when resolving violated global constraints. Based on the theory of ESPC, we propose a *partition-and-resolve procedure*. The procedure iterates between calling a basic solver to solve the constraint-partitioned subproblems, and using a constraint-reweighting strategy to resolve the violated global constraints across the subproblems.

c) We have developed a leading planner based on the constraint-partitioning approach.

We have observed clustered constraint structures in many real-world planning applications and have proposed effective partitioning strategies that exploit the constraint structure. We have also studied specific techniques for planning, including landmark analysis, subgoal ordering, search space pruning, and producible resource detection, that lead to further decomposition of a problem and faster solution of subproblems. Our planner has won the first prize in the suboptimal temporal metric track and the second prize in the suboptimal propositional track of the 4th International Planning Competition (IPC4) in 2004, and is the only planner that has won two prizes in the competition. We have also improved the efficiency of NASA's space-rover planning system by up to 1000 times.

d) We have successfully applied the constraint partitioning approach on large-scale mathematical programming benchmarks. We have proposed automated partitioning strategies for selecting the partitioning dimension and for determining the optimal number of partitions in order to minimize the overall search time. The proposed method has achieved a significant reduction in solution time, and has solved some large-scale mixed-integer and continuous constrained nonlinear optimization problems not solvable before.

1.5 Thesis outline

This thesis is organized as follows.

In Chapter 2, we review previous work. We first review existing penalty methods for constrained optimization, discuss their assumptions and limitations, and explain why existing penalty methods cannot efficiently support constraint partitioning. Second, we review other existing mathematical programming methods, with a focus on existing partitioning methods. We compare different classes of partitioning methods, discuss their limitations, and point out what is new in our partitioning approach. Finally, we survey existing planning algorithms,

and show that most existing methods solve a planning problem as a whole without constraint partitioning.

In Chapter 3, we present our theoretical foundation for constraint partitioning. After introducing the basic concepts, including mixed-space neighborhood and mixed-space constrained local minimum, we present the main theorem, the extended saddle point condition (ESPC), that states a one-to-one correspondence between a constrained local minimum and an extended saddle point defined on a penalty function. We prove the theorem and explain the significance of the result. Next, we extend the ESPC condition to the MINLPs under constraint partitioning. After formulating the MINLP problem under constraint partitioning in a mathematical form, we develop the concepts of neighborhood and constrained local minimum under constraint partitioning, and present the partitioned ESPC condition. Finally, we present an iterative search framework for finding points that satisfy the partitioned ESPC condition, and discuss the global convergence of this search scheme.

In Chapter 4, we study the application of the proposed approach on automated planning. In particular, we study two planning models, including PDDL2.2 planning and ASEPN planning for NASA. For each of the planning models, we demonstrate the constraint structure of the problem, the automated partitioning strategy, the strategy for resolving global constraints, and the implementation details of the planning systems. We also show experimental results of our planning systems on the Third and Fourth International Planning Competitions, and NASA's benchmarks for planning.

In Chapter 5, we present the application on large-scale mathematical programming benchmarks, including both mixed-integer problems and continuous problems. We first describe the test problem sets and demonstrate the constraint structure of the problems. We then discuss an automated partitioning strategy to select the partitioning dimension and the optimal number of partitions. We also examine strategies for updating penalty values in

resolving violated global constraints. We then compare the performance of our solver with other leading solves on these continuous and mixed-integer benchmarks.

Finally, in Chapter 6, we briefly summarize the research work we have presented in this thesis, and point out future directions to extend this research.

Chapter 2

Previous Work

In this chapter, we first review existing penalty methods for solving constrained optimization problems and discuss their assumptions and limitations. We then review existing partitioning methods for solving mathematical programming problems, and point out what is new in our proposed constraint partitioning approach. Finally, we survey existing automated planning algorithms and show that most of them solve a problem as a whole without partitioning.

2.1 Existing Penalty Methods

In this section, we survey existing penalty methods for constrained optimization. The concepts of saddle points and penalty formulations are important and form the basis of our theory presented in Chapter 3.

Penalty methods belong to a general approach that can solve continuous, discrete, and mixed constrained optimization problems, with no continuity, differentiability, and convexity requirements. Penalty methods are the primary methods for solving constrained problems.

A penalty function of P_m is a summation of its objective and its constraint functions weighted by penalties. Using penalty vectors $\alpha \in R^m$ and $\beta \in R^r$, the general penalty function for P_m is:

$$L_p(z, \alpha, \beta) = f(z) + \alpha^T P(h(z)) + \beta^T Q(g(z)), \tag{2.1}$$

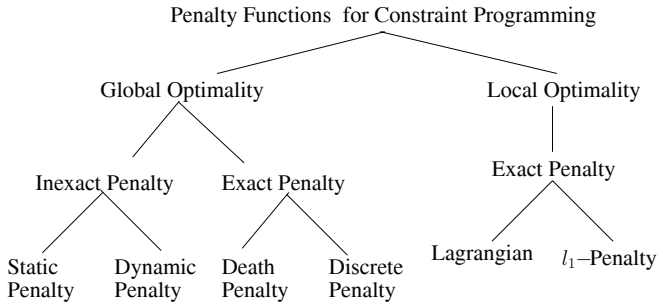


Figure 2.1: A classification of existing penalty methods.

where P and Q are transformation functions. The goal of a penalty method is to find suitable α^* and β^* in such a way that x^* that minimizes (2.1) corresponds to either a constrained global minimum (CGM) that is feasible and has the best objective value in the entire search space, or a constrained local minimum (CLM) that is feasible and has the best objective value in a pre-defined neighborhood. Penalty methods belong to a general approach that can solve continuous, discrete, and mixed constrained optimization problems, with no continuity, differentiability, and convexity requirements.

We show a classification of existing penalty methods in Figure 2.1. Penalty methods can be classified into global optimal penalty methods that look for CGM solutions of P_m and local optimal penalty methods that look for CLM solutions of P_m. By another dimension, penalty methods can be classified into exact penalty methods that can find exact CGM or CLM points under finite penalty values, and inexact penalty methods in which the minimization of a penalty function does not lead to exact CGM or CLM points [10, 58, 9, 70]. Instead, successive minimizations of an inexact penalty function with increasing penalty values lead to points infinitely close to a CGM or CLM solution. Inexact penalty methods converge to a CGM or CLM solution as the penalty values approach infinity.

2.1.1 Global optimal penalty methods

A *static-penalty method* [58, 70] formulates P_m as the minimization of (2.1) when the constraints of P_m are transformed by P and Q with the following properties: a) $P(h(z)) \geq 0$ and $Q(g(z)) \geq 0$; and b) $P(h(z)) = 0$ iff $h(z) = 0$, and $Q(g(z)) = 0$ iff $g(z) \leq 0$. With penalty vectors α and β, an example static-penalty method solves the following problem with constant $\rho \geq 1$:

$$\min_x L_s(z, \alpha, \beta) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i \, |h_i(z)|^\rho + \sum_{j=1}^{r} \beta_j \left(\max(0, g_j(z)) \right)^\rho \right]. \tag{2.2}$$

A static penalty method can be an exact or inexact penalty method, depending on the value of ρ. It is an exact penalty method when $\rho = 1$ and an inexact penalty method when when $\rho > 1$ [10, 58, 70]. That is, when $\rho = 1$, there exist finite penalty values α and β such that the point minimizing the penalty function is exactly the CGM of P_m. However, when $\rho > 1$, the static penalty method is an inexact method and will converge to the CGM as the penalty values approach infinity. We illustrate this property by an example.

Example 1. Consider the following simple optimization problem:

$$\min_x \quad f(x) = x \tag{2.3}$$

$$\text{subject to :} \quad h(x) = x = 10 \tag{2.4}$$

Obviously, the CGM is at $x^* = 10$. The static penalty function for this problem is:

$$L_p(x, \alpha) = f(x) + \alpha |h(x)|^\rho = x + \alpha |x - 10|^\rho \tag{2.5}$$

We consider two cases:

a) When $\rho = 1$, the penalty function becomes:

$$L_p(x, \alpha) = f(x) + \alpha |h(x)|^\rho = x + \alpha |x - 10| \tag{2.6}$$

we can show that there exists a *finite* penalty value $\alpha^* = 1$ such that for any penalty values satisfying:

$$\alpha^{**} \geq \alpha^* = 1, \tag{2.7}$$

the point that minimizes $L_p(x, \alpha^{**})$ is exactly at the CGM $x^* = 10$. To see this, we need to show that:

$$L_p(x, \alpha^{**}) \geq L_p(x^*, \alpha^{**}) = L_p(10, \alpha^{**}) = 10, \forall x \in \mathcal{R}. \tag{2.8}$$

We show that (2.8) is true for both $x > 10$ and $x < 10$. When $x > 10$, we have:

$$L_p(x, \alpha^{**}) = x + \alpha^{**}(x - 10) > x > 10, \qquad \forall x > 10 \ ; \tag{2.9}$$

When $x < 10$, we have:

$$\begin{aligned} L_p(x, \alpha^{**}) &= x + \alpha^{**}(10 - x) = 10\alpha^{**} - (\alpha^{**} - 1)x \\ &= (\alpha^{**} - 1)(10 - x) + 10 \geq 10, \qquad \forall x < 10, \end{aligned} \tag{2.10}$$

where the last inequality is true because $\alpha^{**} \geq \alpha^* = 1$ and $x < 10$.

Therefore, we have shown that $L_p(x, \alpha^{**})$ is minimized exactly at the CGM $x^* = 10$ for all finite penalty values $\alpha^{**} > 1$.

b) When $\rho > 1$, differentiating L_p and setting the result to zero:

$$\frac{d}{dx} L_p(x, \alpha) = 1 + \rho\alpha |x - 10|^{\rho - 1} = 0, \tag{2.11}$$

we have that the minimum of L_p is at:

$$x' = 10 \pm \left(\frac{-1}{\rho\alpha} \right)^{\frac{1}{\rho - 1}}. \tag{2.12}$$

We can see that the static penalty method is an inexact penalty method because x' in (2.12) that minimizes L_p is not exactly at the CGM $x^* = 10$. However, x' will converge to $x^* = 10$ as the penalty value α approaches infinity. ■

A variation of the static-penalty method proposed in [40] uses discrete penalty values and assigns a penalty value $\alpha_i(h_i(z))$ when $h_i(z)$ exceeds a discrete level ℓ_i (*resp.*, $\beta_j(g(z))$ when $\max(0, g(z))$ exceeds a discrete level ℓ_j), where a higher level of constraint violation entails a larger penalty value. The penalty method then solves the following problem:

$$\min_z L_s(z, \alpha, \beta) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(h_i(z))\, h_i^2(z) + \sum_{j=1}^{r} \beta_j(g_j(z)) \Big(\max(0, g_j(z)) \Big)^2 \right], \tag{2.13}$$

It is an inexact penalty method and requires the setting of many parameters. A limitation common to all static-penalty methods is that it is generally very difficult to set the suitable penalty values statically. Also, these methods were developed for finding CGM and cannot relate a CLM of P_m to a local minimum of the corresponding penalty function. Therefore, they are computationally expensive because they involve finding a global minimum of a

nonlinear penalty function. Techniques like simulated annealing [49] can be used, although they only achieve global optimality with asymptotic convergence.

Instead of finding the penalty values by trial and error, a *dynamic-penalty method* [58, 70] increases the penalties in (2.2) gradually, finds the global minimum z^* of (2.2) with respect to z for each combination of penalties, and stops when z^* is a feasible solution to P_m. Like the static penalty methods, a dynamic penalty methods can be an exact or inexact method, depending on the value of ρ. Moreover, it has the same limitation as all static-penalty methods because it requires finding global minima of nonlinear functions.

There are many variations of dynamic penalty methods. A well-known one is the *non-stationary method* (NS) [42] that solves a sequence of problems with the following problem in iteration t:

$$\begin{aligned} \min_z L_t(z,\alpha,\beta) &= \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(t)\ |h_i(z)|^\rho + \sum_{j=1}^{r} \beta_j(t) \Big(\max(0, g_j(z)) \Big)^\rho \right], \quad (2.14) \\ \text{where} \quad & \alpha_i(t+1) = \alpha_i(t) + C \cdot |h_i(z(t))|, \\ & \beta_j(t+1) = \beta_j(t) + C \cdot \max(0, g_j(z(t)), \end{aligned}$$

and $C > 0$ and $\rho > 1$ are constant parameters. An advantage of the NS penalty method is that it requires only a few parameters to be tuned.

Another dynamic penalty method is the *adaptive penalty method* (AP) [8] that makes use of a feedback from the search process. AP solves the following problem in iteration t:

$$\min_z L_t(z,\alpha,\beta) = \min_z \left[f(z) + \sum_{i=1}^{m} \alpha_i(t)\ h_i(z)^2 + \sum_{j=1}^{r} \beta_j(t) \Big(\max(0, g_j(z)) \Big)^2 \right], \quad (2.15)$$

where $\alpha_i(t)$ is, respectively, increased, decreased, or left unchanged when the constraint

$h_i(z) = 0$ is respectively, infeasible, feasible, or neither in the last ℓ iterations. That is,

$$\alpha_i(t+1) = \begin{cases} \alpha_i(t)/\lambda_1 & \text{if } h_i(z(i)) = 0 \text{ is feasible in iterations } t-\ell+1, \ldots, t \\ \lambda_2 \cdot \alpha_i(t) & \text{if } h_i(z(i)) = 0 \text{ is infeasible in iterations } t-\ell+1, \ldots, t \\ \alpha_i(t) & \text{otherwise.} \end{cases} \quad (2.16)$$

where ℓ is a positive integer, $\lambda_1, \lambda_2 > 1$ and $\lambda_1 \neq \lambda_2$ in order to avoid cycles in updates. β is updated in a similar fashion.

The *threshold dynamic penalty method* estimates and adjusts dynamically a near-feasible threshold $q_i(t)$ (*resp.* $q'_j(t)$) for each constraint in iteration t. Each threshold indicates a reasonable amount of violation allowed for promising but infeasible points during the solution of the following problem:

$$\min_z L_t(z,\alpha,\beta) = \min_z \left\{ f(z) + \alpha(t) \left[\sum_{i=1}^{m} \left(\frac{h_i(z)}{q_i(t)} \right)^2 + \sum_{j=1}^{r} \left(\frac{\max(0, g_j(z))}{q'_j(t)} \right)^2 \right] \right\}. \quad (2.17)$$

There are two other variations of global penalty methods that are exact methods. The *death penalty method* simply rejects all infeasible individuals [5] using the following penalty

function:

$$L_p(z,\alpha,\beta) = f(z) + \alpha^T P(h(z)) + \beta^T Q(g(z)), \tag{2.18}$$

$$P(h(z)) = \begin{cases} +\infty & \text{if } h(z) \neq 0 \\ 0 & \text{if } h(z) = 0, \end{cases} \tag{2.19}$$

$$Q(g(z)) = \begin{cases} +\infty & \text{if } g(z) > 0 \\ 0 & \text{if } g(z) \leq 0. \end{cases} \tag{2.20}$$

This is an exact penalty method. Given any finite penalty values $\alpha > 0$ and $\beta > 0$, the minimum point of the penalty function must be feasible and must have the minimum objective value, and therefore is exactly the CGM of P_m. Another exact penalty method is the *discrete penalty method* that uses the numbers of violated constraints instead of the degree of violations in the penalty function [52].

In summary, methods for finding the global minimum of (2.2) are of limited practical importance because the search of a global minimum of a nonlinear function is very computationally expensive. Global optimization techniques like simulated annealing are too slow because they only achieve global optimality with asymptotic convergence.

2.1.2 Local optimal penalty methods

To avoid expensive global optimization, *local optimal penalty methods* have been developed to look for *constrained local minima* (CLM) instead of CGM. These include Lagrange-multiplier methods and ℓ_1-penalty methods, which are both exact penalty methods. They are designed

for solving continuous nonlinear programming problems (CNLPs) defined as follows:

$$(P_c): \quad \min_x \quad f(x) \quad \text{where } x = (x_1,\ldots,x_v)^T \in \mathcal{R}^v \tag{2.21}$$
$$\text{subject to} \quad h(x) = (h_1(x),\ldots,h_m(x))^T = 0 \quad \text{and} \quad g(x) = (g_1(x),\ldots,g_r(x))^T \leq 0,$$

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

Definition 2.1 *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 $f(x^*) \leq f(x)$ for all feasible $x \in \mathcal{N}_c(x^*)$.*

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

$$L(x,\lambda,\mu) = f(x) + \lambda^T h(x) + \mu^T g(x). \tag{2.22}$$

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* [10]. Assuming x^* is a CLM_c and a regular point,[1] then there exist unique $\lambda^* \in \mathcal{R}^m$ and $\mu^* \in \mathcal{R}^r$ such that:

$$\nabla_x L(x^*,\lambda^*,\mu^*) = 0, \tag{2.23}$$

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

[1] Point x is a *regular point* [58] if gradient vectors of equality constraints $\nabla h_1(x),\ldots,\nabla h_m(x)$ and active inequality constraints $\nabla g_{a_1}(x),\ldots,\nabla g_{a_l}(x), a_i \in A(x)$ (the set of active constraints), are linearly independent.

otherwise.

The unique x, λ and μ that satisfy (2.23) can be found by solving (2.23) as a system of nonlinear equations. For instance, consider P_c with only equality constraints. The KKT condition in (2.23) can be expressed as a system of $v+m$ equations in $v+m$ unknowns:

$$F(x,\lambda) = \begin{bmatrix} \nabla f(x) + \lambda^T \nabla h(x) \\ h(x) \end{bmatrix} = 0, \tag{2.24}$$

where $\nabla h(x)^T = [\nabla h_1(x), \ldots, \nabla h_m(x)]$ is the Jacobian of the constraints. The $v+m$ unknowns are solvable when the matrix in (2.24) is nonsingular.

Because the necessary KKT condition is a system of simultaneous nonlinear equations that cannot be solved in closed form, iterative procedures have been developed to find the unique x^* and Lagrange-multiplier values that satisfy the condition. For example, existing sequential quadratic-programming solvers like SNOPT [32] solve the nonlinear equations iteratively by forming a quadratic approximation, evaluating the quadratic model, and updating estimates of x and Lagrange multipliers until a solution has been found. Such an iterative process cannot be used when a problem is partitioned by its constraints into subproblems. Partitioning the constraints amounts to decomposing the system of nonlinear equations into parts and solving each independently before resolving inconsistencies. There is no known procedure for solving a system of partitioned nonlinear equations efficiently when the result requires a unique assignment of each variable. Moreover, the approach is limited to solving CNLPs with continuous and differentiable functions and cannot be applied to solve discrete and mixed-integer problems, the limitation is due to the fact that the existence of the Lagrange multipliers depends on the existence of the gradients of constraint and objective functions and the regularity conditions (independence of constraint gradients)

at the solution points.

b) Sufficient *saddle-point condition* [51, 4]. 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 $\lambda^* \in \mathcal{R}^m$ and $\mu^* \in \mathcal{R}^r$ that satisfy the following saddle-point condition at x^*:

$$L(x^*, \lambda, \mu) \leq L(x^*, \lambda^*, \mu^*) \leq L(x, \lambda^*, \mu^*) \tag{2.25}$$

for all $x \in \mathcal{N}_c(x^*)$ 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 λ^* and μ^* that satisfy (2.25) at each CLM_c x^* of P_c.

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

$$\min_x \quad f(x) = -x^2 \quad \text{subject to} \quad h(x) = x - 5 = 0. \tag{2.26}$$

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 (2.25) to be satisfied at $x^* = 5$.

In practice, it is difficult to use (2.25) for finding the unique x^*, λ^*, and μ^* that satisfy (2.23) 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 (2.23).

Another local optimal exact penalty method for solving CNLPs is the ℓ_1-penalty method

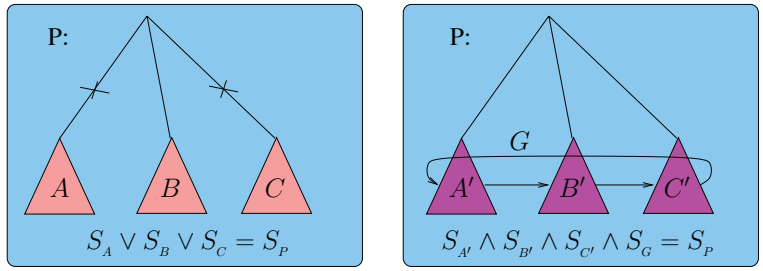
based on the following ℓ_1-penalty function [34]:

$$\ell_1(z,c) = f(z) + c \cdot \max\Big(0, |h_1(z)|, \cdots, |h_m(z)|, g_1(z), \cdots, g_q(z)\Big) \tag{2.27}$$

Its theory shows that there is a one-to-one correspondence between a CLM_c and an unconstrained local minimum of (2.27) when c is larger than a finite threshold c^*. The method cannot support constraint partitioning of (1.1) for two reasons. First, the theory was derived under the continuity and differentiability assumptions on constraints similar to those in the first-order KKT condition. In fact, c^* can be proved to be the maximum of all Lagrange multipliers of the corresponding Lagrangian formulation. Second, since there is only one single penalty term c on the maximum of all constraint violations in (2.27), it is difficult to partition (2.27) by its constraints and to reach a consistent value of c across the subproblems.

In short, using global optimal penalty formulation (2.2) that converts all constraint functions to non-negative functions, a CGM_m of P_m always corresponds to an unconstrained global minimum of (2.2) when its penalties are larger than some thresholds. Unfortunately, this result is impractical because finding global minima of an unconstrained nonlinear function is computationally expensive. On the other hand, using penalty formulation (2.22), a constrained local minimum of the original problem does not imply a local minimum of (2.22) at z^* because there may not exist feasible penalties there. This means that the CLMs whose penalties do not exist in (2.22) cannot be found by looking for local minima of (2.22). Moreover, the Lagrange methods and ℓ_1-penalty methods work for continuous and differentiable problems only.

To cope with these shortcomings, in the next chapter, we propose an exact local optimal penalty method based on an ℓ_1-penalty function and prove that a constrained local minimum of a general MINLP always corresponds to a saddle point of the corresponding ℓ_1-penalty



a) Subspace partitioning b) Constraint partitioning

Figure 2.2: An illustration of subspace partitioning and constraint partitioning. Subspace partitioning decomposes P into a disjunction ($\vee$) of subproblems, where the complexity of each subproblem is similar to that of P. In contrast, constraint partitioning decomposes P into a conjunction ($\wedge$) of subproblems and a set of global constraints (G) to be resolved, where the complexity of each subproblem is substantially smaller than that of P.

function when its penalties are larger than some thresholds. A constrained local minimum of a MINLP can, therefore, be found by looking for a local minimum of the corresponding unconstrained penalty function using an existing algorithm and by increasing gradually the penalties of violated constraints.

2.2 Existing Partitioning Methods for Mathematical Programming

Partitioning has been used in many existing methods for solving NLPs. Partitioning methods can be classified as subspace partitioning methods and constraint partitioning methods. We illustrate a comparison of these two approaches in Figure 2.2.

Subspace partitioning decomposes a problem by partitioning its variable space into sub-

sets and by examining the subspaces one at a time until the problem is solved (Figure 2.2a). Although pruning and ordering strategies can make the search more efficient by not requiring the search of every subspace, the complexity of searching each subspace is very similar to that of the original. In contrast, constraint partitioning decomposes the constraints of a problem into subproblems. Each subproblem is typically much more relaxed than the original and requires significantly less time to solve (see Figure 1.4 for examples). However, there are global constraints (S_G in Figure 2.2b) that may not be satisfied after solving each subproblem independently. As a result, the subproblems may need to be solved multiple times in order to resolve any violated global constraints. The number of times that the subproblems are to be solved depends strongly on the difficulty in resolving the violated global constraints.

2.2.1 Subspace partitioning methods

Many MINLP solution methods are based on subspace partitioning and solve the subproblems obtained by decomposing the search space of a problem instance. MINLP methods generally apply subspace partitioning to decompose the search space of a MINLP problem into 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.

To review the existing methods, we rewrite the MINLP problem P_m in (1.1) below:

$$\begin{aligned} (P_m): \qquad & \min_{x,y} \quad f(x,y), \\ & \text{subject to} \quad h(x,y)=0 \ \text{ and } \ g(x,y)\leq 0, \end{aligned} \tag{2.28}$$

where $x \in \mathcal{R}^v$ are continuous variables, and $y \in \mathcal{D}^w$ are integer variables.

a) *Generalized Benders decomposition* (GBD) [28] 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.

Specifically, GBD alternates between a local phase and a global phase. The local phase solves an NLP subproblem with all integer variables y fixed:

$$\begin{aligned} (NLP_k): \qquad & \min_{x} \quad f(x,y^k), \\ & \text{subject to} \quad h(x,y^k)=0 \\ & \text{and} \quad g(x,y^k)\leq 0, \end{aligned} \tag{2.29}$$

where $x \in \mathcal{R}^v$ are continuous variables, and y^k are fixed integer assignments to y at iteration k. The solutions of the NLP subproblem are feasible solutions to the original MINLP and provides an upper bound to the solution objective.

The global phase of GBD solves a mixed-integer linear programming (MILP) master problem derived from the duality theory and linear approximation to predict a new lower bound of f for the MINLP problem and generate new integer value assignments for y. The search terminates when the predicted lower bound equals the current upper bound. GBD requires the continuous subspace to be a compact and convex set, and the objective and constraint functions to be *convex* and *differentiable*.

GBD is a subspace-partitioning method because it partitions the mixed-integer variable space by fixing the values of integer variables. It explores multiple subspaces by defining different values for integer variables at each iteration. Promising subspaces (integer value assignments) are found by solving a master problem based on a linear approximation.

b) *Outer approximation (OA)* [20] is similar to GBD and solves subproblems with fixed

integer variables. The main difference between OA and GBD is that the master problem of OA is formulated using primal information and outer linearization, while the master problem of GBD is formulated using a dual representation. It requires the continuous subspace to be a nonempty, compact and convex set, and the objective and constraint functions to be convex. Similar to GBD, OA is a subspace-partitioning method.

c) *Branch and bound methods* [75, 76] solve MINLPs by performing a tree enumeration in which a subset of integer variables are successively fixed at each node of the tree. For each node, it relaxes the integrality requirements on the unfixed integer values, forms a continuous NLP subproblem, and solves the continuous NLP subproblem. The solution of the corresponding NLP at each node provides a lower bound for the optimal MINLP objective function value. In addition, if the integer variables in the solution to a NLP subproblem take on integer values, an upper bound is also obtained at the node. The branch and bound method keeps track of the best lower and upper bounds and stops when the two bounds meet. A node is pruned when its lower bound exceeds the current upper bound or when its subproblem is infeasible. A popular MINLP solver MINLP_BB [54] implements branch and bound and uses the sequential quadratic programming (SQP) algorithm to solve the continuous NLP subproblem for each node of the tree search.

One major approach to get the lower bound in branch and bound is *Lagrangian relaxation* [29, 31, 27, 78, 7]. Lagrangian relaxation reformulates P_m into a dual problem. Based on the duality theory [29, 31], the optimal solution to the dual problem has an objective value less than or equal to the objective value of the optimal solution to the original problem. Therefore, a Lagrangian relaxation can derive a lower bound for each node in branch and bound. It should be noted that when the problem is linear or convex, Lagrangian relaxation can be used directly to find an optimal primal solution when given an optimal dual solution, or vice versa. However, as pointed out in [81], it does not work well for nonlinear problems.

Branch and bound is also a subspace-partitioning method. Each subproblem explores a subset of the search space by fixing the value of some integer variables. It uses lower and upper bounds to eliminate those subspaces where no solution is contained. Many of the range-reduction techniques in branch and bound are applicable only when the relaxed problems are convex.

d) *Branch and reduce* [76] is a general search framework implemented in the BARON solver [76]. The branch and reduce optimization framework is a variation of branch and bound that partitions the subspace by branching on the values of discrete variables. The key difference between branch-and-reduce and branch-and-bound is in the derivation of lower bounds for subproblems.

Instead of using conventional techniques, such as linear approximation and Lagrangian relaxation, BARON employs a variety of duality-based and interval arithmetic-based range contraction techniques to derive lower bounds of subproblems. These techniques are specialized modules designed for problems with special properties, such as bilinear programming, fixed-charge programming, indefinite quadratic programming, linear multiplicative programming, separable concave programs, and univariate polynomial programming. These specialized modules can derive tighter lower bounds than general conventional methods and lead to more effective pruning of the subspaces [76]. Like branch and bound, branch and reduce is also a subspace partitioning method.

e) *Generalized cross decomposition (GCD)* [38, 39, 74] is a variation of GBD that iterates between a phase solving the primal and dual subproblems and a phase solving the master problem. It differs from GBD only in the definition of the master problem. Similar to OA and GBD, GCD requires the objective and constraint functions of subproblems to be proper convex functions.

f) *Extended Cutting Plane (ECP)* is a recent method for solving convex MINLPs [93]

directly without partitioning. Unlike GBD, OA, and branch and bound, ECP does not rely on the use of NLP subproblem and algorithms. Instead, it relies on the iterative solution of a series of approximated linear problems based on linearization of some constraint functions. In each step, it cuts the feasible region by adding a linearization of the most violated constraint at the current point. The ECP method requires the objective function to be linear and the constraint functions to be convex.

g) *Direct-solution methods* attack a problem without any transformation or partitioning. Examples include reject/discarding methods [41, 6, 73, 67], repair methods [43, 63], feasible-direction methods [9, 53, 58], preserving feasibility methods [59], and strategic oscillation methods [33, 77]. These methods try to limit the probes in the feasible region or transforming them into feasible points by some repair operators. 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 subspace partitioning. They are not applicable for solving general MINLPs in (1.1) due to their restricted requirements on the decomposed subproblems. All these methods require the MINLP problem to have some special properties, such as nonempty and compact subspaces with convex objective and constraint functions.

2.2.2 Separable programming methods

Another class of decomposition methods are *separable programming methods* based on duality [10]. An extensive introduction to separable programming can be found in Chapters 5 and 6 of the nonlinear programming book by Bertsekas [10]. Dual methods solve a dual problem instead of the primal problem by finding Lagrange multipliers to maximize the dual function, instead of minimizing the Lagrangian function in the original variable subspace. If the problem has some separable properties, maximizing the dual function can be decomposed

into multiple much simpler subproblems, each involving only a subset of the constraints and variables.

A typical problem that can be solved by separable programming has the following form, where variables x has m components $x_1, \cdots, x_m$ of dimension $n_1, \cdots, n_m$, respectively:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} f_i(x_i) \\ \text{subject to} \quad & \sum_{i=1}^{m} g_{ij} \leq 0, j = 1, \cdots, r, \\ & x_i \in X_i, i = 1, \cdots, m. \end{aligned} \tag{2.30}$$

Here f_i and g_{ij} are given continuous and differentiable functions, and X_i is a given subset in R^{n_i}. Note that if the constraints $\sum_{i=1}^{m} g_{ij} \leq 0$ were not present in (2.30), then it would be straightforward to decompose this problem into m independent subproblems. However, the constraints link all the subproblems together and create possibly global inconsistencies.

Separable programming methods consider the following dual problem of (2.30):

$$\text{maximize} \quad q(\mu) \tag{2.31}$$

$$\text{subject to} \quad \mu \geq 0, \tag{2.32}$$

where μ is the vector of Lagrange multipliers and the dual function $q(\mu)$ is formulated as:

$$q(\mu) = \inf_{x_i \in X_i, i=1..m} \left\{ \sum_{i=1}^{m} \left(f_i(x_i) + \sum_{j=1}^{r} \mu_j g_{ij}(x_i) \right) \right\} = \sum_{i=1}^{m} q_i(\mu) \tag{2.33}$$

and

$$q_i(\mu) = \inf_{x_i \in X_i} \left\{ f_i(x_i) + \sum_{j=1}^{r} \mu_j g_{ij}(x_i) \right\}, \quad i = 1, \cdots, m. \tag{2.34}$$

Therefore, the minimization involved in computing the dual function $q(\mu)$ in (2.33) can be decomposed into m simpler subproblems in (2.34). These minimizations on the subproblems can be done efficiently when the functions in the subproblems are convex or linear, which lead to efficient computation of the overall dual function.

Separable programming has similar advantages as our constraint partitioning in that it decomposes a large problem into multiple much simpler subproblems so that the total time to solve all subproblems is usually much less than the time to solve the original problem. However, There are several limitations of separable programming. First, it requires the objective and constraint functions to have a separable structure shown in (2.30). Second, separable programming methods have restricted assumptions, such as linearity or convexity of the functions, that limit their general applications. An example is the Danzig-Wolfe decomposition [18] that works for problems with separable objective functions and linear constraints. Third, this method is only necessary but not sufficient in the sense that it does not guarantee finding the solution due to the duality gap. That is, the solution of the dual problem may not be a solution to the original problem, and there is a gap between the maximum value of the dual function and the minimum value of the original objective function. In general, there is no duality gap for convex problems and linear problems, and there is a non-zero duality gap for general nonlinear problems [10].

In this research, we study general constrained optimization with no restricted assumptions on the constraint functions. Instead of using duality, we build our theoretical foundation on a penalty formulation discussed in the next section. We show that our condition is necessary

and sufficient, and there is a one-to-one correspondence between the points satisfying our condition and the local optimal points.

2.2.3 Remarks on existing partitioning methods

Partitioning has been used in many existing methods for solving NLPs. Partitioning methods can be classified as subspace partitioning methods and constraint partitioning methods. Subspace partitioning decomposes a problem by partitioning its variable space into subspaces and by evaluating the subspaces individually. The total complexity of enumerating all subspaces is very similar to that of searching the original space. Most existing MINLP methods, including GBD, OA, branch and bound, branch and reduce, GCD, and ECP are subspace partitioning methods.

In contrast, the constraint partitioning approach decomposes the constraints of a problem into subproblems. Each subproblem is typically much more relaxed than the original and requires significantly less time to solve. Therefore, the total time to solve all subproblems is significantly reduced. Separable programming is a constraint partitioning method that works for linear and convex NLPs with a separable structure. In this thesis, we study a general constraint partitioning approach that works for general NLPs without special assumptions on functions and structure.

2.3 Existing Planning Algorithms

Planning is the problem of generating a course of actions to finish a give task, subject to propositional, temporal, and numerical constraints. A planning problem involves a time horizon for actions to take place and a state space to represent the internal configurations. Figure 2.3 classifies existing planning and scheduling methods based on their state and temporal representations and the search techniques used.

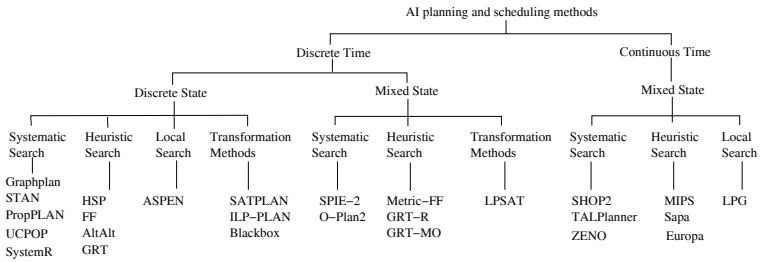


Figure 2.3: A classification of existing planning and scheduling approaches.

2.3.1 Discrete-time discrete-state methods

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 UCPOP [64], an early goal-directed planner based on the Partial Order Causal Link (POCL) technique; Graphplan [11], that searches a planning graph in order to minimize the length of a parallel plan; STAN [57], an efficient implementation of Graphplan; PropPLAN [25], a planner based on a naive breadth-first search of ordered binary decision diagrams; and System R [56], a method based on regression that solves one goal at a time.

Heuristic solvers explore a partitioned subspace represented as a complete planning problem. Within a subproblem, local searches employ guidance heuristics evaluated over the entire temporal horizon in estimating the distance from a state to the goal state. They are not guaranteed to find feasible plans because their success depends on the guidance heuristics used. Examples include HSP [12], a hill-climbing search using heuristic values obtained by solving a relaxed problem; FF [37], an enforced hill-climbing search using heuristic values obtained by solving a relaxed Graphplan problem; AltAlt [62], a hybrid planner on top of

STAN and HSP; GRT [71] (and its extension to MO-GRT [72]), a two-phase planner that first estimates the distances between domain facts and goals, before searching by a simple best-first strategy; and ASPEN [16], a repair-based local-search method that can handle discrete temporal and metric constraints and that optimizes multiple objectives in the form of a weighted sum.

Last, transformation methods convert a problem into a constrained optimization or satisfaction problem, before solving it by an existing solver based on subspace partitioning. Examples in this class include SATPLAN [46] Blackbox [47], and ILP-PLAN [48].

2.3.2 Discrete-time mixed-state methods

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 generally use subspace partitioning in their solution process. SIPE-2 [94] and O-Plan2 [80] are early Hierarchical Task Network (HTN) planners. The HTN planners suffer the deficiencies of domain dependency, difficulty in engineering, and brittleness in handling unexpected states. Metric-FF [37] employs heuristic searches similar to those in FF, using a modified heuristic function to accommodate numeric constraints and to favor the optimization of a given cost function. GRT-R [71] is an extension of GRT for solving problems whose resources are represented numerically. Last, LPSAT [96] uses the LCNF representation by combining propositional logic and linear equalities and inequalities, and searches by a SAT solver which calls an LP system.

2.3.3 Continuous-time mixed-state methods

Continuous-time mixed-state methods can be classified into systematic, heuristic, and local searches. Again, they rely on subspace partitioning in their solution process. Examples in-

clude LPG [30], a local search guided by a function weighted by discrete penalties on an action graph and a temporal precedence graph; MIPS [21], an A^*-search planner that employs static analysis, numeric estimation, plan relaxation, and critical path analysis to derive heuristic functions with embedded optimization measures; Sapa [79], a heuristic-search planner that employs distance-based heuristics to control its search and that adjusts its heuristics to account for resource constraints and optimization objectives; ZENO [65], a systematic POCL planner with goal-directed planning that can handle continuous time and metric quantities; SHOP2 [60], an HTN planner that searches by problem reduction using a domain-specific knowledge base of methods; TALplanner [19], a logic-based forward-chaining planner using domain-dependent search-control knowledge represented as formulas in Temporal Action Logic (TAL); and Europa [44], a general framework that uses a constraint-based interval (CBI) representation for representing plan constraint network and in its heuristic search.

2.3.4 Remarks on existing planning methods

From the perspective of partitioning, most general and popular methods for solving large planning problems, such as systematic search, heuristic search, and transformation methods, can be viewed as an approach that partitions recursively a search space into independent subproblems by branching on the values of its variables, and that solves each subproblem individually until a feasible solution is found. Since the approach results in subproblems whose aggregate complexity is the same as that of the original problem, it is often combined with intelligent backtracking that employs variable/value ordering to order the subproblems generated, that pre-filters partial inconsistent assignments to eliminate infeasible subproblems, and that prunes subproblems using bounds computed by relaxation or approximation.

Subspace partitioning can be applied directly or indirectly for solving planning problems. Direct methods include complete and heuristic searches. As illustrated in Figure 2.4, these

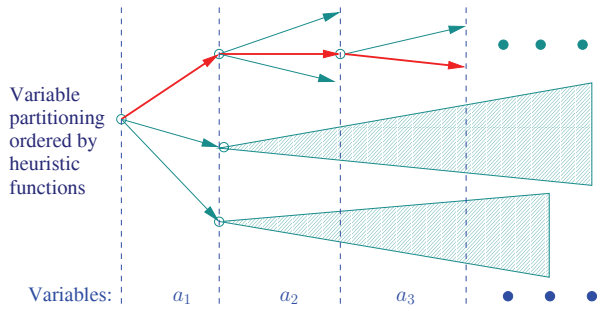


Figure 2.4: Subspace partitioning in planning by branching on variable assignments.

methods partition recursively a search space by branching on the assignment of variables (selection of actions). Their difference is that a complete search enumerates systematically all subspaces, whereas a heuristic search orders and prioritizes subspaces based on heuristic functions. In contrast, in indirect partitioning, a planning problem is first transformed into a satisfiability or an optimization problem, before the transformed problem is partitioned by its search space into subproblems.

In this thesis, we propose an orthogonal *constraint-partitioning approach* that decomposes the constraints of a planning problem into subproblems, each with local constraints and is related to other subproblems by global constraints. Our approach is based on the strong locality of constraints observed in many planning applications. We present in Chapter 4 our partition-and-resolve planning strategy that iterates between solving the subproblems using a basic planner, while considering the local constraints and an objective biased by the global-constraint violations, and using a constraint-reweighting strategy that resolves violated global constraints across the subproblems

In summary, existing planners solve a problem as a whole without partitioning, or apply subspace partitioning to decompose a problem into subproblems, or transform a problem into another form before solving it by existing methods based on subspace partitioning. In this

thesis, 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 to solve the small subproblems, using an existing planner is more effective because it saves a lot of development efforts. We develop in Chapter 4 our approach on planning problems.

Chapter 3

Theory of Extended Saddle Points

In this chapter, we propose a complete theory to characterize the constrained local optimal solutions of NLPs in discrete, continuous, and mixed spaces. Based on a penalty formulation, our theory offers a necessary and sufficient condition for constrained local optima. The unique feature of this condition is that, the condition is true over an extended region of penalty values, instead of a set of unique values.

We then extend the theory and apply it to NLPs under constraint partitioning and derive a set of partitioned necessary conditions to reduce the search complexity in revolving global constraints under constraint partitioning. The theory facilitates constraint partitioning by providing a set of necessary conditions, one for each subproblem, to characterize the constrained local optima. It reduces the complexity by defining a much smaller search space in each subproblem for backtracking. It is significantly stronger than the partial filtering based on local constraints alone, as it incorporates violated global constraints and the objective into its local constrained optimality condition.

3.1 Necessary and Sufficient Extended Saddle-Point Condition

We describe in this section our theory of extended saddle points in discrete, continuous, and mixed spaces based on an ℓ_1-penalty function. We show a necessary and sufficient condition under a relaxed range of penalties. Since the result for MINLPs is derived based on the

results for continuous and discrete NLPs, we will first develop the theory for continuous and discrete problems before presenting a unified theory for mixed problems.

3.1.1 ESPC for continuous optimization

We first state the necessary and sufficient ESPC on CLM_c of P_c in (2.21), based on an ℓ_1-penalty function that transforms the constraints of P_c into non-negative functions.

Definition 3.1 The ℓ_1-penalty function for P_c in (2.21) is defined as follows:

$$L_c(x, \alpha, \beta) = f(x) + \alpha^T |h(x)| + \beta^T \max(0, g(x)), \tag{3.1}$$

where $|h(x)| = (|h_1(x)|, \ldots, |h_m(x)|)^T$ and $\max(0, g(x)) = (\max(0, g_1(x)), \ldots, \max(0, g_r(x)))^T$; and $\alpha \in \mathcal{R}^m$ and $\beta \in \mathcal{R}^r$ are penalty vectors.

In continuous space, 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.

Definition 3.2 $D_x(\phi(x'), \vec{p})$, the *subdifferential* 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,

$$D_x(\phi(x'), \vec{p}) = \lim_{\epsilon \to 0} \frac{\phi(x' + \epsilon\vec{p}) - \phi(x')}{\epsilon}. \tag{3.2}$$

Definition 3.3 *Constraint-qualification condition.* Solution $x^* \in X$ of P_c 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,

$$\nexists\, \vec{p} \in X \text{ such that } D_x(h_i(x^*), \vec{p}) = 0 \text{ and } D_x(g_j(x^*), \vec{p}) = 0 \;\; \forall i \in C_h \text{ and } j \in C_g, \tag{3.3}$$

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

Intuitively, constraint qualification at x^* ensures the existence of finite α and β that lead to a local minimum of (3.1) 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, all the active constraints at $x^* + \vec{p}$ are also satisfied, and it will be impossible to find finite α and β in order to establish a local minimum of (3.1) at x^* with respect to $x^* + \vec{p}$. To ensure a local minimum of (3.1) at x^*, the above scenario must not be true for any $\vec{p}$ at x^*.

Our constraint-qualification condition requires the subdifferential of at least one active constraint at x^* to be non-zero along each and every direction $\vec{p}$. For CNLPs, the condition rules out the case in which there exists a direction $\vec{p}$ at x^* along which all active constraints are continuous and have zero subdifferentials. Note that our condition is less restricted than the regularity condition in KKT that requires the linear independence of gradients of active constraint functions.

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

Theorem 3.1 *Necessary and sufficient ESPC on CLM_c of P_c.* Suppose $x^* \in \mathcal{R}^v$ is a point in the continuous search space of P_c and satisfies the constraint-qualification condition (3.3), then x^* is a CLM_c of P_c if and only if there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that the following is satisfied:

$$L_c(x^*, \alpha, \beta) \;\leq\; L_c(x^*, \alpha^{**}, \beta^{**}) \;\leq\; L_c(x, \alpha^{**}, \beta^{**}) \quad \text{where } \alpha^{**} > \alpha^* \geq 0 \text{ and } \beta^{**} > \beta^* \geq 0 \tag{3.4}$$

for all $x \in \mathcal{N}_c(x^*)$, $\alpha \in \mathcal{R}^m$, and $\beta \in \mathcal{R}^r$. Here, $\alpha^{**} > \alpha^*$ (*resp.* $\beta^{**} > \beta^*$) means that each

element of α^{**} (*resp.* β^{**}) is larger than the corresponding element of α^* (*resp.* β^*).

Proof. The proof consists of two parts.

"$\Rightarrow$" part: Given x^*, we need to prove that there exist finite $\alpha^{**} > \alpha^* \geq 0$ and $\beta^{**} > \beta^* \geq 0$ that satisfy (3.4). The inequality on the left of (3.4) is true for all α and β because x^* is a CLM_c, which implies that $|h(x^*)| = 0$ and $\max(0, g(x^*)) = 0$.

To prove the inequality on the right of (3.4), we prove for any $x \in \mathcal{N}_c(x^*)$ that there exist finite α^* and β^* such that the inequality is satisfied for any $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$. Let $x = x^* + \epsilon\overrightarrow{p}$, where $\|\overrightarrow{p}\| = 1$ is a unit directional vector and 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, (3.4) implies $f(x) \geq f(x^*)$ and, regardless the choice of the penalties,

$$L_c(x, \alpha^{**}, \beta^{**}) = f(x) \geq f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \tag{3.5}$$

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

$$|h_k(x)| > \xi > 0 = h_k(x^*). \tag{3.6}$$

The above must be true because $h_k(x)$ would be continuous along $\overrightarrow{p}$ if (3.6) were false.

If we set $\alpha_k^{**} > \alpha_k^* = 1$ and when ϵ is small enough, then from (3.6):

$$\begin{aligned} L_c(x, \alpha^{**}, \beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**}|h_i(x)| + \sum_{j=1}^{r} \beta_j^{**} \max(0, g_j(x)) \\ &\geq f(x) + \alpha_k^{**}|h_k(x)| \quad > \quad f(x^*) + \epsilon \nabla_x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \alpha_k^* \xi \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \end{aligned} \tag{3.7}$$

3) If there exists a discontinuous active inequality constraint g_k along $\overrightarrow{p}$, then for a small enough ϵ, there exists a finite positive ξ such that:

$$\max(0, g_k(x)) > \xi > 0. \tag{3.8}$$

If we set $\beta_k^{**} > \beta_k^* = 1$ and when ϵ is small enough, then from (3.8):

$$\begin{aligned} L_c(x, \alpha^{**}, \beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**}|h_i(x)| + \sum_{j=1}^{r} \beta_j^{**} \max(0, g_j(x)) \\ &\geq f(x) + \beta_k^{**} \max(0, g_k(x)) \quad > \quad f(x^*) + \epsilon \nabla_x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \beta_k^* \xi \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}). \end{aligned} \tag{3.9}$$

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

$$\begin{aligned} L_c(x, \alpha^{**}, \beta^{**}) &= f(x) + \sum_{i=1}^{m} \alpha_i^{**}|h_i(x)| + \sum_{j=1}^{r} \beta_j^{**} \max(0, g_j(x)) \\ &\geq f(x) + \alpha_k^{**}|h_k(x)| \quad \geq \quad f(x^*) + \epsilon \nabla_x f(x^*)^T \overrightarrow{p} + o(\epsilon^2) + \alpha_k^{**} \epsilon |D_x(h_k(x^*), \overrightarrow{p})| \\ &\geq f(x^*) + \epsilon \left(\alpha_k^{**} \left| D_x(h_k(x^*), \overrightarrow{p}) \right| - \left| \nabla_x f(x^*)^T \overrightarrow{p} \right| \right) + o(\epsilon^2) \\ &> f(x^*) = L_c(x^*, \alpha^{**}, \beta^{**}) \end{aligned} \tag{3.10}$$

The inequality on the right of (3.4) is proved after combining Cases (1) to (4).

"⇐" part: Assuming (3.4) is satisfied, we need to prove that x^* is a CLM_c. Point x^* is feasible because the inequality on the left of (3.4) can only be satisfied when $h(x^*) = 0$ and $g(x^*) \leq 0$. Since $|h(x^*)| = 0$ and $\max(0, g(x^*)) = 0$, the inequality on the right of (3.4) ensures that x^* is a local minimum when compared to all feasible points in $\mathcal{N}_c(x^*)$. Therefore, x^* is a CLM_c. ■

Intuitively, (3.4) shows that a local minimum of (3.1) with respect to x corresponds to a CLM_c of P_c (second inequality of (3.4)) when α^{**} and β^{**} are larger than some thresholds α^* and β^* such that all the constraints of P_c are forced to be satisfied (first inequality of (3.4)). Since a local minimum of (3.1) can be found easily by many existing search algorithms, our result improves over the static-penalty approach, which is defined with respect to difficult-to-find global minima of (2.2). We can find a CLM_c to P_c by gradually increasing α^{**} and β^{**}, while minimizing $L_c(x, \alpha^{**}, \beta^{**})$, until $\alpha^{**} > \alpha^*$ and $\beta^{**} > \beta^*$.

It is interesting to note that α^* and β^* can be much smaller than the corresponding c^* in the static and dynamic penalty methods. Continuing on the example in (2.26), static and dynamic penalty methods will require $c^{**} > c^* = 1005$ in order to have a global minimum of $L_s(x, c^{**})$ at $x^* = 5$ for $-1000 \leq x \leq 1000$. In contrast, it suffices to have $\alpha^{**} > \alpha^* = 10$ in order to have a local minimum of $L_c(x, \alpha^{**}) = -x^2 + \alpha^{**}|x - 5|$ at $x^* = 5$, irrespective of the range of x. Figure 3.1 illustrates that $L_c(x, \alpha^{**})$ is 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 that makes it easier for global search algorithms to locate local minima.

3.1.2 ESPC for discrete optimization

Next, we present the ESPC of *discrete nonlinear programming* (DNLP) problems. This part is developed by Wah and Wu in 1999 [91, 98]. We sketch the results here, and the complete

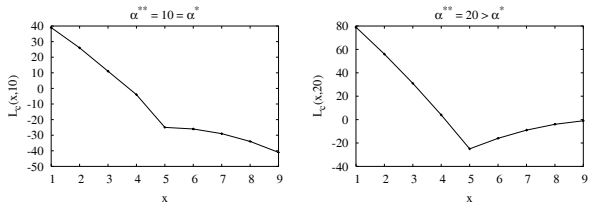


Figure 3.1: An illustration that (3.4) is satisfied when $\alpha^{**} > \alpha^* = 10$ for the CNLP problem in (2.26). $L_c(x, \alpha^{**})$ is a strict local minimum around $x^* = 5$ when $\alpha^{**} > \alpha^*$ but is not one when $\alpha^{**} = \alpha^*$.

theory can be found in the original papers [91, 98].

Consider the DNLP whose f, g and h are not necessarily continuous and differentiable with respect to y.

$$(P_d): \quad \min_y \quad f(y) \quad \text{where } y = (y_1, \ldots, y_w)^T \in \mathcal{D}^w \qquad (3.11)$$
$$\text{subject to} \quad h(y) = 0 \quad \text{and} \quad g(y) \leq 0.$$

The goal of solving P_d is to find a constrained local minimum y^* with respect to $\mathcal{N}(y^*)$, 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, $\mathcal{N}(y)$ represents points that are perturbed from y, with no requirement that there be valid state transitions from y.

Definition 3.4 Discrete neighborhood $\mathcal{N}(y)$ [1] of $y \in \mathcal{D}^w$ in discrete space is a *finite* user-defined set of points $\{y' \in \mathcal{D}^w\}$ such that y' is reachable from y in one step, that $y' \in \mathcal{N}(y) \Longleftrightarrow y \in \mathcal{N}(y')$, and that it is possible to reach every y'' from any y in one or more steps through neighboring points.

Definition 3.5 Point y^* is a CLM_d, a constrained local minimum of P_d with respect to

points in $\mathcal{N}(y^*)$, if y^* is feasible and $f(y^*) \leq f(y)$ for all feasible $y \in \mathcal{N}(y^*)$.

There are two distinct features of CLM_d. First, the set of CLM_d of P_d are 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 a greedy search.

Definition 3.6 The ℓ_1-penalty function for P_d is defined as follows:

$$L_d(y, \alpha, \beta) = f(y) + \alpha^T |h(y)| + \beta^T \max(0, g(y)) \quad \text{where } \alpha \in \mathcal{R}^m \text{ and } \beta \in \mathcal{R}^r. \tag{3.12}$$

Theorem 3.2 *Necessary and sufficient ESPC on CLM_d of P_d* [98, 91]. Suppose $y^* \in \mathcal{D}^w$ is a point in the discrete search space of P_d. Then y^* is a CLM_d of P_d if and only if there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ such that the following is satisfied for all $y \in \mathcal{N}(y^*)$, $\alpha \in \mathcal{R}^m$, and $\beta \in \mathcal{R}^r$:

$$L_d(y^*, \alpha, \beta) \leq L_d(y^*, \alpha^{**}, \beta^{**}) \leq L_d(y, \alpha^{**}, \beta^{**}) \quad \text{where } \alpha^{**} > \alpha^* \geq 0 \text{ and } \beta^{**} > \beta^* \geq 0 \tag{3.13}$$

Proof. The original proof of this theorem is in Zhe Wu's doctorate dissertation [98]. We sketch the idea here. The proof consists of two parts.

"$\Rightarrow$" part: Given y^*, we need to prove that there exist finite $\alpha^{**} > \alpha^* \geq 0$ and $\beta^{**} > \beta^* \geq 0$ that satisfy (3.13). 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 $y \in \mathcal{N}(y^*)$, and let the initial $\alpha^* = \beta^* = 0$. For every y such that $|h(y)| > 0$ (*resp.* $\max(0, g(y)) > 0$), there is at least one constraint that is not satisfied. For each such constraint, we update its penalty to make it large enough in order to offset the possible improvement in the objective value. This update is repeated for every violated constraint of P_d and every $y \in \mathcal{N}(y^*)$ until no further update is possible. Since $\mathcal{N}(y^*)$ has a finite number of elements in discrete space, the update will terminate in finite time and result in finite α^* and β^* values that satisfy (3.13).

"$\Leftarrow$" part: Assuming (3.13) is satisfied, we need to prove that y^* is a CLM_d. The proof is straightforward and is similar to that in the proof of Theorem 3.1. ■

Note that the constraint-qualification condition in Theorem 3.1 is not needed in Theorem 3.2 because constraint functions are not changing continuously in discrete problems.

3.1.3 ESPC for mixed optimization

Last, we present the ESPC for MINLP problems defined in (1.1).

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 thesis, 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 theory for the two subspaces to be developed separately. Because a discrete neighborhood is user-defined and a mixed neighborhood is a union of discrete and continuous neighborhoods, a mixed neighborhood is also a user-defined concept.

Definition 3.7 Mixed neighborhood $\mathcal{N}_m(x, y)$ of $(x, y) \in \mathcal{R}^v \times \mathcal{D}^w$ in mixed space is made up of the union of the continuous neighborhood and the user-defined discrete neighborhood:

$$\mathcal{N}_m(x,y) = \mathcal{N}_c(x)\big|_y \ \cup \ \mathcal{N}(y)\big|_x = \Big\{(x',y) \mid x' \in \mathcal{N}_c(x)\Big\} \bigcup \Big\{(x,y') \mid y' \in \mathcal{N}(y)\Big\}. \quad (3.14)$$

Definition 3.8 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 $f(x^*, y^*) \leq f(x, y)$ for all feasible $(x, y) \in \mathcal{N}_m(x^*, y^*)$.

Definition 3.9 The ℓ_1-penalty function of P_m is defined as follows:

$$L_m(x, y, \alpha, \beta) = f(x, y) + \alpha^T |h(x, y)| + \beta^T \max(0, g(x, y)) \quad \text{where } \alpha \in \mathcal{R}^m \text{ and } \beta \in \mathcal{R}^r \quad (3.15)$$

Theorem 3.3 *Necessary and sufficient ESPC on CLM_m of P_m.* Suppose $(x^*, y^*) \in \mathcal{R}^v \times \mathcal{D}^w$ is a point in the mixed search space of P_m, and x^* satisfies the constraint qualification condition in (3.3) for given y^*, 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 condition is satisfied for all $(x, y) \in \mathcal{N}_m(x^*, y^*)$ and all $\alpha \in \mathcal{R}^m$ and $\beta \in \mathcal{R}^r$:

$$L_m(x^*, y^*, \alpha, \beta) \leq L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y, \alpha^{**}, \beta^{**}) \quad (3.16)$$

$$\text{where} \qquad \alpha^{**} > \alpha^* \geq 0 \text{ and } \beta^{**} > \beta^* \geq 0.$$

Proof. The proof consists of two parts.

"$\Rightarrow$" part: Given (x^*, y^*), we need to prove that there exist finite $\alpha^* \geq 0$ and $\beta^* \geq 0$ so that $(x^*, y^*, \alpha^{**}, \beta^{**})$ satisfy (3.16). The first inequality in (3.16) 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 (3.16), we know that fixing y at y^* converts P_m into

P_c. Further, from Theorem 3.1, there exist finite α_c^* and β_c^* such that:

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y^*, \alpha^{**}, \beta^{**}), \quad \forall\ x \in \mathcal{N}_c(x^*)\big|_{y^*}, \quad (3.17)$$

$$\alpha^{**} > \alpha_c^* \geq 0, \text{ and } \beta^{**} > \beta_c^* \geq 0.$$

Similarly, fixing x at x^* converts P_m into P_d. Hence, from Theorem 3.2, we know that there exist finite α_d^* and β_d^* such that for the same α^{**} and β^{**} in (3.17):

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x^*, y, \alpha^{**}, \beta^{**}), \quad \forall\ y \in \mathcal{N}(y^*)\big|_{x^*}, \quad (3.18)$$

$$\alpha^{**} > \alpha_d^* \geq 0, \text{ and } \beta^{**} > \beta_d^* \geq 0.$$

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

$$\alpha^* = \max(\ \alpha_c^*,\ \alpha_d^*\ ) = [\ \max(\alpha_{c_1}^*, \alpha_{d_1}^*), \ldots, \max(\alpha_{c_m}^*, \alpha_{d_m}^*)\ ]^T \quad (3.19)$$

$$\beta^* = \max(\ \beta_c^*,\ \beta_d^*\ ) = [\ \max(\beta_{c_1}^*, \beta_{d_1}^*), \ldots, \max(\beta_{c_r}^*, \beta_{d_r}^*)\ ]^T, \quad (3.20)$$

we conclude, based on (3.17) and (3.18), that the second inequality in (3.16) is satisfied for all $(x, y) \in \mathcal{N}_m(x^*, y^*)$ and any $\alpha^{**} > \alpha^* \geq 0$ and $\beta^{**} > \beta^* \geq 0$.

"$\Leftarrow$" part: Assuming (3.16) is satisfied, we need to prove that (x^*, y^*) is a CLM_m. The proof is straightforward and is similar to that in the proof of Theorem 3.1. ■

The following theorem facilitates the search of points that satisfy (3.16) by partitioning the condition into two independent necessary conditions. It follows directly from (3.14), which defines $\mathcal{N}_m(x, y)$ to be the union of points perturbed in either the discrete or the continuous subspace. Such partitioning cannot be accomplished if a mixed neighborhood like $\mathcal{N}_c(x) \times \mathcal{N}(y)$ were used.

Theorem 3.4 Given the definition of $\mathcal{N}_m(x, y)$ in (3.14), the ESPC in (3.16) can be rewrit-

ten into two necessary conditions that, collectively, are sufficient:

$$L_m(x^*, y^*, \alpha, \beta) \leq L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x^*, y, \alpha^{**}, \beta^{**}) \quad \text{where} \quad y \in \mathcal{N}(y^*)\big|_{x^*} \tag{3.21}$$

$$L_m(x^*, y^*, \alpha^{**}, \beta^{**}) \leq L_m(x, y^*, \alpha^{**}, \beta^{**}) \quad \text{where} \quad x \in \mathcal{N}_c(x^*)\big|_{y^*} \tag{3.22}$$

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.

The ℓ_1-penalty function is different from the traditional Lagrangian function and the ℓ_1-penalty function discussed in Chapter 2. Unlike the Lagrangian function which uses the original constraint function and requires exact penalty values, our formulation transforms each constraint function into a nonnegative function and does not require exact penalty values. Because ESPC does not require unique penalty values in satisfying Theorem 3.3, the search can be carried out in a partitioned fashion in which we solve each subproblem by looking for penalty values that are larger than the ones required by the original solution. This is not possible if the search were formulated as the solution of a system of nonlinear equations. Second, unlike the ℓ_1-penalty function in (2.27) that has a single penalty term c, there are multiple penalty terms in the penalty function that allow the condition to be partitioned. Moreover, unlike the ℓ_1-penalty theory that requires continuity and differentiability, Theorem 3.3 was developed for general constraint functions that are not necessarily continuous and differentiable.

ESPC overcomes some deficiencies of previous work. Unlike previous global penalty

methods that require expensive global optimization, ESPC provides a condition for locating constrained local optima, which leads to much lower complexity. Unlike the KKT condition that works only for continuous and differentiable problems, ESPC offers a uniform treatment to problems defined in continuous, discrete, and mixed spaces, and does not require the functions to be differentiable or in closed form. Moreover, the unique Lagrange-multiplier values in KKT are typically found by solving a system of nonlinear equations iteratively. The solution process is not applicable to constraint partitioning, because it updates all variables and Lagrange multipliers in each iteration and cannot be carried out in a partition-and-resolve approach. In contrast, ESP supports partitioned searches by allowing an extended region of penalty values. Another advantage of constraint partitioning based on ESPC is that it leads to subproblems of similar nature but of a smaller scale, and any existing solver can be used to solve the subproblems. This feature extends the applicability of our approach.

3.1.4 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 Figure 3.2a. (The algorithm for solving 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 (3.4), 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^{**} (*resp.* β^{**}) is larger than its maximum bound $\bar{\alpha}$ (*resp.* $\bar{\beta}$), where $\bar{\alpha}$ (*resp.* $\bar{\beta}$) is chosen to be so large that it exceeds α^* (*resp.* β^*).

Figure 3.2b shows the pseudo code which solves P_m by looking for x^*, y^*, α^{**}, and

$\alpha \longrightarrow 0$; $\beta \longrightarrow 0$;
repeat
  increase α_i by δ **if** $(h_i(x) \neq 0$ **and** $\alpha_i < \bar{\alpha}_i)$ **for** $i = 1, \ldots, m$;
  increase β_j by δ **if** $(g_j(x) \not\leq 0$ **and** $\beta_j < \bar{\beta}_j)$ **for** $j = 1, \ldots, r$;
  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 $(\alpha_i > \bar{\alpha}_i$ for all $h_i(x) \neq 0$ **and** $\beta_j > \bar{\beta}_j$ for all $g_j(x) \not\leq 0)$
  or a CLM_c of P_c is found.

a) Direct implementation of (3.4) to look for CLM_c of P_c.

$\alpha \longrightarrow 0$; $\beta \longrightarrow 0$;
repeat
  increase α_i by δ **if** $(h_i(x) \neq 0$ **and** $\alpha_i < \bar{\alpha}_i)$ **for** $i = 1, \ldots, m$;
  increase β_j by δ **if** $(g_j(x) \not\leq 0$ **and** $\beta_j < \bar{\beta}_j)$ **for** $j = 1, \ldots, r$;
  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 is 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 is found;
until $(\alpha_i > \bar{\alpha}_i$ for all $h_i(x) \neq 0$ **and** $\beta_j > \bar{\beta}_j$ for all $g_j(x) \not\leq 0)$
  or a CLM_m of P_m is found.

b) Direct implementation of (3.21) and (3.22) to look for CLM_m of P_m

Figure 3.2: Iterative procedures to look for CLM_c of P_c and CLM_m of P_m.

β^{**} that satisfy Theorem 3.4. 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 $(x', y') \in \mathcal{N}_m(x, y)$. The outer loop increases the penalties of violated constraints and stops when a CLM_m is found or when α^{**} (*resp.* β^{**}) exceeds its maximum bound $\bar{\alpha}$ (*resp.* $\bar{\beta}$).

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 Figure 3.2 to terminate without finding a constrained local minimum. The following theorem summarizes this observation.

Theorem 3.5 When $\bar{\alpha} > \alpha^*$ and $\bar{\beta} > \beta^*$, the iterative procedure in Figure 3.2a (*resp.*

Figure 3.2b) generates fixed points that are necessary but not sufficient to satisfy (3.4) (*resp.* (3.21) and (3.22)).

To cope with this issue, we discuss some additional strategies in the procedure to look for CLM_m. These procedures 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 by 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, because functions in some applications may not be in closed form and their gradients are unavailable, it is hard to locate local minima of the ℓ_1-penalty function in this case. To cope with this issue, probes can be generated based on deterministic, probabilistic, or genetic mechanisms and be accepted based on deterministic or stochastic criteria. For example, in our experiments on SGPlan_t(ASPEN) (Section 4.2.1), new probes generated using ASPEN's built-in mechanism during the descent of the ℓ_1-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 (CSA) algorithm [89], 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.

3.2 ESPC under Constraint Partitioning

In this section, we provide a formal definition of MINLPs under constraint partitioning and the related definitions of partitioned neighborhood and constrained local optima. By applying ESPC to the partitioned problem, we decompose the condition into a set of necessary conditions that collectively are sufficient.

3.2.1 Basic definitions for partitioned MINLPs

Consider P_t, a version of (1.1) whose constraints can be partitioned into $N+1$ stages. Stage t, $t = 0, \ldots, N$, of P_t has local *state vector* $z(t) = (z_1(t), \ldots, z_{u_t}(t))^T$, where $z(t)$ 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 formulation of P_t is as follows:

$$
\begin{aligned}
(P_t): \qquad & \min_{z} \quad J(z) && && (3.23)\\
& \text{subject to} \quad h^{(t)}(z(t)) = 0, \quad g^{(t)}(z(t)) \le 0 && \text{(local constraints)}\\
& \text{and} \quad H(z) = 0, \qquad G(z) \le 0 && \text{(global constraints).}
\end{aligned}
$$

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 functions in stage t that involve $z(t)$; and $H = (H_1, \ldots, H_p)^T$ and $G = (G_1, \ldots, G_q)^T$ are global-constraint functions that involve $z \in X \times Y$, the original variables. We assume that J is continuous and differentiable with respect to its continuous variables, that f is lower bounded, and that g and h are general functions that are not necessarily continuous and differentiable and that

can be unbounded.

In this section, we solve P_t in (3.23) by finding solution z that is a CLM_m with respect to feasible solutions in its mixed neighborhood $\mathcal{N}_m(z)$. After showing that z satisfies the ESPC in (3.16), we decompose the ESPC into a set of necessary conditions that collectively are sufficient. P_t is then solved by finding an extended saddle point in each stage and by resolving those violated global constraints using appropriate penalties.

To simplify our discussion, we do not partition solution 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 Theorem 3.4. To enable the partitioning of the ESPC into independent necessary conditions, we define a mixed neighborhood of solution z as follows:

Definition 3.10 $\mathcal{N}_b(z)$, the *mixed neighborhood of* z for a partitioned problem, is defined as:

$$
\mathcal{N}_b(z) = \bigcup_{t=0}^{N} \mathcal{N}_p^{(t)}(z) = \bigcup_{t=0}^{N} \left\{ z' \;\middle|\; z'(t) \in \mathcal{N}_m(z(t)) \text{ and } \forall z_i \notin z(t),\ z_i' = z_i \right\}, \tag{3.24}
$$

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

Intuitively, $\mathcal{N}_b(z)$ is separated into $N+1$ neighborhoods, each perturbing z in only one of the stages of P_t, while keeping the overlapped variables consistent in other stages. The size of $\mathcal{N}_b(z)$ defined in (3.24) is smaller than the Cartesian product of the neighborhoods across all stages.

3.2.2 Necessary and sufficient ESPC for partitioned subproblems

By considering P_t as a MINLP and by defining the corresponding ℓ_1-penalty function, we can apply Theorem 3.3 as follows.

Definition 3.11 Let $\Phi(z,\gamma,\eta) = \gamma^T|H(z)| + \eta^T \max(0, G(z))$ be the sum of the transformed global constraint functions weighted by their penalties, where $\gamma = (\gamma_1, \ldots, \gamma_p)^T \in \mathcal{R}^p$ and $\eta = (\eta_1, \ldots, \eta_q)^T \in \mathcal{R}^q$ are the penalty vectors for the global constraints. Then the ℓ_1-penalty function for P_t in (3.23) and the corresponding ℓ_1-penalty function in stage t are defined as follows:

$$L_m(z,\alpha,\beta,\gamma,\eta) = J(z) + \sum_{t=0}^{N}\left\{\alpha(t)^T|h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t))\right\} + \Phi(z,\gamma,\eta), \quad (3.25)$$

$$\Gamma_d(z,\alpha(t),\beta(t),\gamma,\eta) = J(z) + \alpha(t)^T|h^{(t)}(z(t))| + \beta(t)^T \max(0, g^{(t)}(z(t))) + \Phi(z,\gamma,\eta) \quad (3.26)$$

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 the penalty vectors for the local constraints in stage t.

Lemma 3.1 Plan z is a CLM_m of (3.23) with respect to $\mathcal{N}_b(z)$ if and only if there exist finite nonnegative $\alpha^*, \beta^*, \gamma^*$ and η^* such that the following ESPC is satisfied:

$$L_m(z^*,\alpha,\beta,\gamma,\eta) \le L_m(z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}) \le L_m(z,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}), \quad (3.27)$$
$$\text{where } \alpha^{**} > \alpha^* \ge 0,\ \beta^{**} > \beta^* \ge 0,\ \gamma^{**} > \gamma^* \ge 0,\ \text{and } \eta^{**} > \eta^* \ge 0,$$

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 $z \in \mathcal{N}_b(z^*)$.

Based on Lemma 3.2.2, we next show the partitioning of (3.27) into multiple conditions.

Theorem 3.6 *Partitioned necessary and sufficient ESPC on CLM_m of P_t.* Given $\mathcal{N}_b(z)$, the ESPC in (3.27) can be rewritten into $N+2$ necessary conditions that, collectively, are sufficient:

$$\Gamma_d(z^*,\alpha(t),\beta(t),\gamma^{**},\eta^{**}) \le \Gamma_d(z^*,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**}) \le \Gamma_d(z,\alpha(t)^{**},\beta(t)^{**},\gamma^{**},\eta^{**}) \quad (3.28)$$

$$L_m(z^*,\alpha^{**},\beta^{**},\gamma,\eta) \le L_m(z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}) \quad (3.29)$$

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

Proof. We prove the theorem by showing the equivalence of (3.27) and the combined (3.28) and (3.29).

"⇒" part: Given z^* that satisfies (3.27), we show that it also satisfies (3.28) and (3.29). Since for all $t = 0, \ldots, N$, any $z \in \mathcal{N}_p^{(t)}(z^*)$ is also a point in $\mathcal{N}_b(z^*)$; hence, the second inequality in (3.28) is implied by the second inequality in (3.27). The first inequality in (3.28) and the inequality in (3.29) are obvious, as all the constraints are satisfied at z^*.

"⇐" part: We prove this part by contradiction. Assuming that z^* satisfies (3.28) and (3.29) but not (3.27), the first inequality in (3.27) cannot be violated because the first inequality in (3.28) and the inequality in (3.29) imply that all local and global constraints are satisfied. Therefore, it must be the second inequality in (3.27) that is not satisfied at z^*. That is, there exist $z \in \mathcal{N}_b(z^*)$ and a unique t' where $z \in \mathcal{N}_b^{(t')}(z^*)$ (according to the definition of $\mathcal{N}_b(z)$ in (3.24)) such that:

$$L_m(z^*,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}) \not\le L_m(z,\alpha^{**},\beta^{**},\gamma^{**},\eta^{**}). \quad (3.30)$$

This implies that the second inequality in (3.28) is not satisfied at $t = t'$, which contradicts our assumption that z^* satisfies (3.28) and (3.29). Our argument proves that any z^* that satisfies (3.28) and (3.29) must also satisfy (3.27). ■

Theorem 3.6 shows that the original ESPC in Theorem 3.3 can be partitioned into $N+1$ necessary conditions in (3.28) and an overall necessary condition in (3.29) on the global constraints across the subproblems. A close examination shows that local extended saddle points that satisfy (3.28) in Stage t are local minima of (3.26) with respect to z (the second inequality of (3.28)), 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 (3.28)). In essence, points that satisfy (3.28) in Stage t is a solution to the following MINLP, whose

objective is biased by the violated global constraints:

$$\min_{z(t)} \quad J(z) + \gamma^T |H(z)| + \eta^T \max(0, G(z)) \tag{3.31}$$

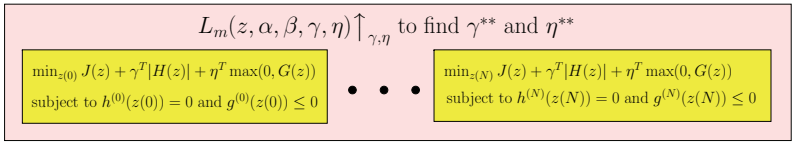
$$\text{subject to} \quad h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \leq 0.$$

The bias on the violated global constraints when solving (3.31) 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 (3.31) will lead to points, if they exist, that satisfy the global constraints.

In short, finding points that satisfy (3.27) can be reduced to solving multiple MINLPs, where the problem in Stage t defined by (3.31) can be handled easily by an existing solver, and to the reweighting of violated global constraints defined in (3.29).

3.2.3 The partition-and-resolve procedure

Figure 5.2 presents the *partition-and-resolve* procedure that looks for points satisfying the conditions in Theorem 3.6. The inner loop of stage t in Figure 5.2b solves (3.31) by looking for an extended saddle point that satisfies (3.28). This can be done by the procedure in Figure 3.2b, using fixed γ and η specified in the outer loop. Another way is to solve (3.31) directly using an existing solver. This is possible because (3.31) is a well-defined MINLP. As is illustrated in Chapters 4 and 5, we can use an existing solver to solve the partitioned planning subproblems defined by (3.31). After solving the subproblems, the penalties on the 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 3.5, the procedure in Figure 5.2 generates fixed points that are necessary but not sufficient to satisfy (3.21) and (3.22). Hence, additional steps described



a) Partitioned search to look for points that satisfy (3.29) and (3.31)

$\gamma \longrightarrow 0$; $\eta \longrightarrow 0$;
repeat // increase the penalties on violated global constraints //
increase γ_i by δ **if** $(H_i(z) \neq 0$ **and** $\gamma_i < \bar{\gamma}_i)$ **for** $i = 1, \ldots, p$;
increase η_j by δ **if** $(G_j(z) \not\leq 0$ **and** $\eta_j < \bar{\eta}_j)$ **for** $j = 1, \ldots, q$;
for $t = 0$ **to** N // iterate over all $N+1$ stages to solve (3.31) in each stage //
apply an existing solver or the procedure in Figure 3.2b to solve (3.31)
end_for;
until $(\gamma_i > \bar{\gamma}_i$ for all $H_i(z) \neq 0$ **and** $\eta_j > \bar{\eta}_j$ for all $G_j(z) \not\leq 0)$ **or** a CLM_m of P_t is found.

b) Implementation for finding CLM_m of P_t that satisfies (3.29) and (3.31)

Figure 3.3: The partition-and-resolve procedure to look for CLM_m of P_t.

in Section 3.1.4 are needed to help escape from infeasible local minima of the ℓ_1-penalty function.

3.3 Asymptotic Convergence of Stochastic Partitioned Search

The partitioned procedure outlined in Figure 3.3 is guaranteed to stop at a fixed point of the ℓ_1- penalty function but may not converge to a CLM. In this section, we proposed a stochastic partitioned-search procedure to look for extended saddle points. The procedure carries out a process of simulated annealing (SA) in the joint space of multiple partitioned subproblems and converges to a constrained global minimum of the original problem.

Simulated annealing (SA) [1, 35] is a method for solving unconstrained optimization problems. It performs stochastic descents of the objective function using a control parameter called temperature. The complete theory for proving the asymptotic convergence of

SA on unconstrained optimization is based on a discrete Markov chain model. The constrained simulated annealing (CSA) [89] algorithm extends SA to solve constrained DNLPs and converges asymptotically with probability one to a constrained global minimum. The CSA algorithm performs stochastic ascents in the original-variable space and descents in the penalty space. In each step, CSA generates a probe in a user-defined neighborhood by perturbing either the original variables or the penalty vector. Each probe is accepted using a Metropolis probability controlled by a temperature. It has been proved that CSA using a logarithmic schedule to reduce temperatures converges asymptotically with probability one to a constrained global minimum [89, 92].

Since the Markov chain model is discrete, our study will be limited to DNLPs formulated in P_d only. Note that the results can be extended to CNLPs and MNLPs when continuous variables are discretized. A detailed analysis of discretization has been studied before [98].

To simplify the presentation, without loss of generality, we study the following equality-constrained DNLP in this section:

$$\begin{aligned} (P_e): \qquad \min_{y} \quad & f(y) \ \text{ where } \ y \in \mathcal{D}^w \\ \text{subject to} \quad & h(y) = 0, \end{aligned} \tag{3.32}$$

where $\mathcal{D}^w$ is the discrete search space.

Our objective is to find the constrained global minimum CGM_d to P_e defined as follows:

Definition 1. A point $y^* \in \mathcal{D}^w$ is a CGM_d to the DNLP in P_e if and only if $h(y^*) = 0$ and $f(y^*) \leq f(y)\ \forall y \in \mathcal{D}^w$. We define Y_{opt} to be the set of all CGM_d of P_e: $Y_{opt} = \{y^* | y^* \text{ is a } CGM_d \text{ of } P_e\}$.

Suppose the constraints of P_e are partitioned into $N+1$ stages as follows:

$$\begin{aligned} (P_{et}): \qquad \min_{y} \quad & J(y) \\ \text{subject to} \quad & h^{(t)}(y(t)) = 0, \qquad \text{(local constraints)} \\ \text{and} \quad & H(y) = 0, \qquad \text{(global constraints)}, \end{aligned} \tag{3.33}$$

where stage t, $t = 0, \ldots, N$, has local *state vector* $y(t) = (y_1(t), \ldots, y_{u_t}(t))^T$, and $y(t)$ includes all the variables that appear in any of the local constraints in stage t.

The ℓ_1-penalty function for P_e under constraint partitioning is as follows:

$$L(y, \alpha, \gamma) = f(y) + \sum_{t=0}^{N} \alpha(t) |h^{(t)}(y(t))| + \gamma |H(y)| \tag{3.34}$$

From Theorem 3.6, the subproblem we need to solve in each stage t, $t = 0, \cdots, N$, is defined as follows:

$$\begin{aligned} P_{sub}^{(t)}: \qquad \min_{y(t)} \quad & J(y) + \gamma^T |H(y)| \\ \text{subject to} \quad & h^{(t)}(y(t)) = 0 \end{aligned} \tag{3.35}$$

3.3.1 Constraint Partitioned Simulated Annealing (CPSA) Algorithm

The goal of constrained partitioned simulated annealing (CPSA) is to find an extended saddle point with the minimum objective value, i.e. a CGM_d, of P_{et}, by solving the set of partitioned subproblems and by resolving the violated global constraints.

The CPSA algorithm is based on the framework in Figure 3.3. Instead of considering all

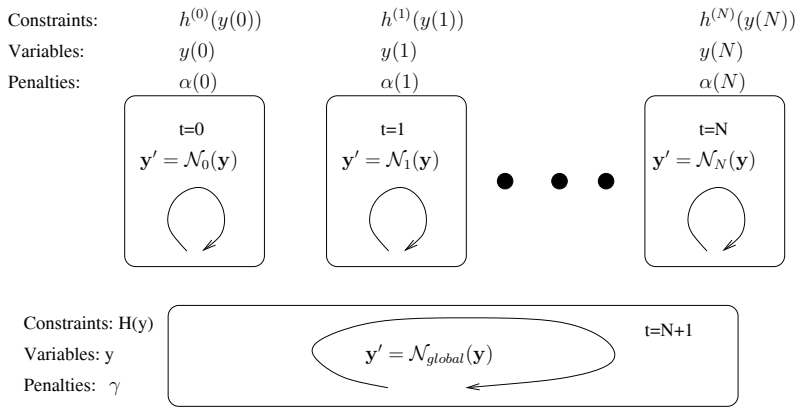


Figure 3.4: The search procedure of constraint partitioned simulated annealing (CPSA).

the constraints together without partitioning as in CSA, CPSA performs searches in multiple subproblems, each with a small subset of constraints.

Before presenting the CPSA procedure, we illustrate its idea in Figure 3.4. Unlike the original CSA that probes the whole space and takes all the constraints into account in each step, CPSA partitions the constraints into $N+1$ subsets. For subproblem $t, t = 0, ..N$, we still perform constrained simulated annealing, but only in the smaller space of those variables relevant to the local constraints $h^{(t)}(y(t))$, including the original variables $y(t)$ and the penalties $\alpha(t)$. In addition, there is a global search that perturbs the original variable y and the penalties γ for the global constraints. This additional search component is needed in order to resolve any violated global constraint in $H(y)$ and ensure the asymptotic convergence of CPSA to CGM_d.

Figure 3.5 describes the CPSA procedure. It begins from starting point $\mathbf{y} = (y, \alpha, \gamma)$ (Line 2), where y can be either user-provided or randomly generated, and $\alpha = \lambda = 0$.


```
1.  procedure Constraint_Partitioned_Simulated_Annealing (CPSA)
2.      set starting values of y = (y, α, γ);
3.      set starting temperature T = T^0 and cooling rate 0 < ξ < 1;
4.      set N_T (number of trials per temperature);
5.      while stopping condition is not satisfied do
6.          for k ← 1 to N_T do
7.              set subproblem t to be a random integer from 0 to N + 1;
8.              if 0 ≤ t ≤ N
9.                  generate trial point y' from N_t(y) using q_t(y, y');
10.                 accept y' with probability A_T(y, y');
11.             else /* t = N + 1 */
12.                 generate trial point y' from N_global(y) using q_global(y, y');
13.                 accept y' with probability A_T(y, y');
14.             end_if
15.         end_for
16.         reduce temperature by T ⟵ ξ · T;
17.     end_while
18. end_procedure
```

Figure 3.5: CPSA: the constraint partitioned simulated annealing algorithm.

The initial temperature T^0 is set (Line 3) to be so large that almost all trial points $\mathbf{y}'$ will be accepted. Line 4 sets N_T, the number of iterations at each temperature. At each fixed temperature, we generate N_T probes. In each step, we randomly pick a subproblem t between 0 and $N+1$. $p_t(i)$, the probability that $t = i$, can be chosen arbitrarily as long as:

$$\sum_{i=0}^{N+1} p_t(i) = 1 \quad \text{and} \quad p_t(i) > 0. \tag{3.36}$$

When t is an integer between 0 to N, we generate a probe in the search space of subproblem t that perturbs $y(t)$ and $\alpha(t)$ (Line 9); when $t = N+1$, it is a global exploration step where we generate a probe that perturbs y and γ in the global constraints (Line 12).

Line 9 generates a random trial point $\mathbf{y}'$ in the neighborhood $\mathcal{N}_t(\mathbf{y})$ of the current point

$\mathbf{x} = (y, \alpha, \gamma)$ in search space $S = \mathcal{D}^w \times \mathcal{R}^p \times \mathcal{R}^q$, where $\mathcal{N}_t(\mathbf{y})$ is defined as follows:

$$\begin{aligned}
\mathcal{N}_t(\mathbf{y}) &= \{(y', \alpha, \gamma) \in S \text{ where } y' \in \mathcal{N}_t^1(y)\} \bigcup \{(y, \alpha', \gamma) \in S \text{ where } \alpha' \in \mathcal{N}_t^2(\alpha)\} \\
\mathcal{N}_t^1(y) &= \left\{ y' \;\middle|\; y'(t) \in \mathcal{N}_d(y(t)) \text{ and } \forall y_i \notin y(t),\ y'_i = y_i \right\}, \qquad (3.37) \\
\mathcal{N}_t^2(\alpha) &= \left\{ \alpha' \;\middle|\; \alpha'(t) \in \mathcal{R}^{P(t)} \text{ and } \forall \alpha_i \notin \alpha(t),\ \alpha'_i = \alpha_i \right\}. \qquad (3.38)
\end{aligned}$$

Here, $\mathcal{N}_d(y(t))$ is a user-defined discrete-space neighborhood of variable vector $y(t)$ in stage t. Intuitively, $\mathcal{N}_t(\mathbf{y})$ is the neighborhood of the original variable vector $y(t)$ and the penalty vector $\alpha(t)$ in subproblem t. In solving subproblem t, we generate a point $\mathbf{y}' \in \mathcal{N}_t(\mathbf{y})$ with a generation probability $q_t(\mathbf{y}, \mathbf{y}')$ that can be arbitrarily defined as long as the following condition is satisfied:

$$0 \le q_t(\mathbf{y}, \mathbf{y}') \le 1 \text{ and } \sum_{\mathbf{y}' \in \mathcal{N}_t(\mathbf{y})} q_t(\mathbf{y}, \mathbf{y}') = 1. \qquad (3.39)$$

Line 12 generates a random trial point $\mathbf{y}'$ in the neighborhood $\mathcal{N}_{global}(\mathbf{y})$ of the current point $\mathbf{y} = (y, \alpha, \gamma)$ in search space $S = \mathcal{D}^w \times \mathcal{R}^p \times \mathcal{R}^q$, where $\mathcal{N}_{global}(\mathbf{y})$ is defined as follows:

$$\begin{aligned}
\mathcal{N}_{global}(\mathbf{y}) &= \{(y, \alpha, \gamma) \in S \text{ where } y' \in \mathcal{N}^3(y)\} \bigcup \{(y, \alpha, \gamma') \in S \text{ where } \gamma' \in \mathcal{N}^4(\gamma)\}, \\
\mathcal{N}^3(y) &= \bigcup_{t=0}^{N} \left\{ y' \;\middle|\; y'(t) \in \mathcal{D}^{y(t)} \text{ and } y'_i = y_i\ \forall y_i \notin y(t) \right\}, \qquad (3.40) \\
\mathcal{N}^4(\gamma) &= \left\{ \gamma' \;\middle|\; \gamma' \in \mathcal{R}^q \right\}, \qquad (3.41)
\end{aligned}$$

Intuitively, probes in $\mathcal{N}_{global}(\mathbf{y})$ are made to resolve violated global constraints by updating

γ and to have a global exploration in the y-space by perturbing y. $\mathcal{N}_{global}(\mathbf{y})$ ensures the connectivity of the underlying Markov chain model of the process, which is required to achieve the asymptotic convergence of the algorithm. We will identify where this definition is used in the proof for asymptotical convergence.

We accept $\mathbf{y}'$ according to acceptance probability $A_T(\mathbf{y}, \mathbf{y}')$:

$$A_T(\mathbf{y}, \mathbf{y}') = \begin{cases} exp\left(-\frac{(L(\mathbf{y}')-L(\mathbf{y}))^+}{T}\right) & \text{if } \mathbf{y}' = (y', \alpha, \gamma) \\ exp\left(-\frac{(L(\mathbf{y})-L(\mathbf{y}'))^+}{T}\right) & \text{if } \mathbf{y}' = (y, \alpha', \gamma) \quad \text{or } \mathbf{y}' = (y, \alpha, \gamma') \end{cases} \qquad (3.42)$$

where $(a)^+ = a$ if $a > 0$, and $(a)^+ = 0$ otherwise for all $a \in \mathcal{R}$.

The CPSA algorithm fits in the ESPC search framework in Figure 3.3a by performing ascents in the penalty subspace and descents in the y subspace in order to solve each subproblem individually. However, when compared to the deterministic search algorithm in Figure 3.3b, there are a couple of major differences with CPSA. First, instead of deterministically enumerating the subproblems from 0 to N in a round-robin fashion, CPSA randomly picks a subproblem in each step. We use this random selection in order to have a memory-less Markovian process in CPSA. Since a round-robin selection is not memory-less, the random selection of subproblems is used to facilitate the convergence analysis in a Markov-chain model. Second, instead of deterministic decreases of the penalty function in the y subspace and deterministic increases in the penalty subspaces for α and γ in Figure 3.3b, CPSA accepts new probes in all the three subspaces stochastically using an acceptance probability controlled by a decreasing temperature. This is a technique used in constrained simulated annealing for ensuring global convergence. Based on the above modifications, we can prove in the next section that CPSA converges asymptotically to a CGM_d with probability one.

Comparing to CSA, CPSA reduces the search complexity through constraint partition-

ing. Since both CSA and CPSA need to converge to an equilibrium distribution of variables at a given temperature before the temperature is reduced, the total search time depends on the convergence time at each fixed temperature. By partitioning the constraints into subsets, each subproblem involves an exponentially smaller subspace with only a small number of variables and penalty values. Thus, each subproblem takes significantly less time to converge to an equilibrium distribution at a given temperature, and the total time for all the subproblems to converge is also reduced significantly.

3.3.2 Asymptotic convergence of CPSA

In this section, we prove the asymptotic convergence of CPSA to a CGM_d y^* of P_d by modelling the process as an inhomogeneous Markov chain and by following a similar line as the original proof for the asymptotic convergence of CSA [98]. We show that the Markov chain is strongly ergodic, that the Markov chain minimizes an implicit virtual energy based on the framework of generalized SA (GSA) [82], and that the virtual energy is at the its minimum at any CGM_d.

Inhomogeneous Markov Chain

CPSA can be modelled by an inhomogeneous Markov chain that consists of a sequence of homogeneous Markov chains of finite length, each at a specific temperature in a given temperature schedule. In order to model the CPSA algorithm by a Markov chain, we need a tuple with three components to describe the state space of the Markov chain: $\mathbf{y} = (y, \alpha, \gamma)$, where $y \in \mathcal{D}^w$ is the original variable and α, γ are the penalty vectors.

According to the generation probability $q(\mathbf{y}, \mathbf{y}')$ and acceptance probability $A_T(\mathbf{y}, \mathbf{y}')$,

the *one-step transition probability* of the Markov chain is:

$$P_T(\mathbf{y}, \mathbf{y}') = \begin{cases} p_t(i) q_i(\mathbf{y}, \mathbf{y}') A_T(\mathbf{y}, \mathbf{y}') & \text{if } \mathbf{y}' \in \mathcal{N}_i(\mathbf{y}), i = 0, \cdots, N \\ p_t(N+1) q_{N+1}(\mathbf{y}, \mathbf{y}') A_T(\mathbf{y}, \mathbf{y}') & \text{if } \mathbf{y}' \in \mathcal{N}_{global}(\mathbf{y}) \\ 1 - \sum_{i=0}^{N} \sum_{\mathbf{y} \in \mathcal{N}_i(\mathbf{y})} P_T(\mathbf{y}, \mathbf{y}) - \sum_{\mathbf{y} \in \mathcal{N}_{global}(\mathbf{y})} P_T(\mathbf{y}, \mathbf{y}) & \text{if } \mathbf{y}' = \mathbf{y} \\ 0 & \text{otherwise,} \end{cases} \tag{3.43}$$

and the corresponding *transition matrix* is $P_T = [P_T(\mathbf{y}, \mathbf{y}')]$. Note that when $\mathbf{y}' \in \mathcal{N}_i(\mathbf{y}), i = 0, \cdots, N$, $\mathbf{y}'$ in fact only differs from $\mathbf{y}$ in $y^{(i)}$ of the i^{th} subproblem and remains the same for the other parts. It is different from CSA which perturbs $\mathbf{y}$ in the overall variable space.

Let $\mathbf{y}_{opt} = \{(y^*, \alpha, \gamma) |\ y^* \in Y_{opt}\}$, and N_L be the maximum of the minimum number of transitions required to reach $\mathbf{y}_{opt}$ from all $\mathbf{y}$ in the Markov space. Consider the sequence of temperatures $\{T_k, k = 0, 1, 2, \cdots\}$, where $T_k > T_{k+1}$ and $\lim_{k \to \infty} T_k = 0$, and set N_T, the number of trials per temperature, to be N_L. The following theorem proves the strong ergodicity of the Markov chain.

Theorem 3.7 The inhomogeneous Markov chain is *strongly ergodic* if the sequence of temperatures $\{T_k\}$ satisfy:

$$T_k \geq \frac{N_L \triangle_L}{ln(k+1)}, \tag{3.44}$$

where $\triangle_L = \max_{\mathbf{y}} \{|L(\mathbf{y}') - L(\mathbf{y})|, \mathbf{y}' \in \mathcal{N}(\mathbf{y})\}$.

Proof. The proof of strong ergodicity follows the steps used to show the weak ergodicity of CSA [98] and use the strong ergodicity conclusions [2, 3]. Before we provide the detailed proof, we outline the strategy of the proof and point out its difference from the original proof

of strong ergodicity of CSA [98].

Based on previous strong ergodicity conclusions [2, 3], the key of the proof is to show the following inequality:

$$\sum_{k=0}^{\infty}[1-\tau_1(P_{T_k}^{N_T})] \geq \infty. \tag{3.45}$$

where $P_{T_k}^{N_T}$ is the N_T-step transition matrix of the Markov chain under temperature T_k and $\tau_1(P)$ is known as the coefficient of ergodicity of a transition matrix P. Since the N_T-step transition matrix $P_{T_k}^{N_T}$ is the convolution of the one-step transition matrix P_{T_k}, we show that all entries in the one-step transition matrix P_{T_k} is larger than a lower bound. CSA has derived such a lower bound for P_{T_k}. In this proof, we have derived a different lower bound for P_{T_k} based on the new transition matrix of CPSA.

a) Let:

$$\triangle_G = \min_{\mathbf{y}\in S, \mathbf{y}'\in\mathcal{N}_i(\mathbf{y})|i=0,\cdots,N+1} q_i(\mathbf{y},\mathbf{y}'), \quad \text{and} \quad \triangle_P = \min_{i=0,\cdots,N+1} p_t(i) \tag{3.46}$$

For all $\mathbf{y}\in S$ and $\mathbf{y}'\in\mathcal{N}(\mathbf{y})$, we have

$$P_{T_k}(\mathbf{y},\mathbf{y}') = \sum_{i=0}^{N+1} p_t(i) q_i(\mathbf{y},\mathbf{y}') A_{T_k}(\mathbf{y},\mathbf{y}') \geq \triangle_P \triangle_G \, e^{-\triangle_L/T_k}, \tag{3.47}$$

because $(L(\mathbf{y}')-L(\mathbf{y}))^+ \leq \triangle_L$ for $\mathbf{y}'=(y',\alpha,\gamma)$ and $(L(\mathbf{y})-L(\mathbf{y}'))^+ \leq \triangle_L$ for $\mathbf{y}'=(y,\alpha',\gamma)$ or $\mathbf{y}'=(y,\alpha,\gamma')$, according to the definition of $\triangle_L$.

b) Based on (3.47), for all $\mathbf{y},\mathbf{y}\in S$ and $k\geq k_0$, the N_T-step transition probability from

$\mathbf{y}=\mathbf{y}_0$ to $\mathbf{y}=\mathbf{y}_{N_T}$ satisfies the following:

$$P_{T_k}^{N_T}(\mathbf{y},\mathbf{y}) \geq P_{T_k}(\mathbf{y}_0,\mathbf{y}_1)P_{T_k}(\mathbf{y}_1,\mathbf{y}_2)\cdots P_{T_k}(\mathbf{y}_{N_T-1},\mathbf{y}_{N_T}) \geq \left[\triangle_P\triangle_G e^{-\triangle_L/T_k}\right]^{N_T}.$$

Let $\tau_1(P)$ be the coefficient of ergodicity of transition matrix P. Then the lower bound of $1-\tau_1(P_{T_k}^{N_T})$ is:

$$\begin{aligned}
1-\tau_1(P_{T_k}^{N_T}) &= \min_{\mathbf{y},\mathbf{y}'\in S}\sum_{\mathbf{y}\in S}\min\{P_{T_k}^{N_T}(\mathbf{y},\mathbf{y}), P_{T_k}^{N_T}(\mathbf{y}',\mathbf{y})\} \\
&\geq \min_{\mathbf{y},\mathbf{y}'\in S}\min_{\mathbf{y}'\in S}\{P_{T_k}^{N_T}(\mathbf{y},\mathbf{y}'), P_{T_k}^{N_T}(\mathbf{y}',\mathbf{y}')\} \\
&\geq \left[\triangle_P\triangle_G e^{-\triangle_L/T_k}\right]^{N_T} = \triangle_P^{N_T}\triangle_G^{N_T} e^{-\triangle_L N_T/T_k}.
\end{aligned}$$

Then using any temperature schedule that satisfies (3.44), the following holds:

$$\sum_{k=0}^{\infty}[1-\tau_1(P_{T_k}^{N_T})] \geq \sum_{k=k_0}^{\infty}\triangle_P^{N_T}\triangle_G^{N_T}e^{-\triangle_L N_T/T_k} \geq \triangle_P^{N_T}\triangle_G^{N_T}\sum_{k=k_0}^{\infty}\frac{1}{k+1} = \infty. \tag{3.48}$$

Therefore, the Markov chain is weakly ergodic.

c) In addition, because the transition probability $P_{T_k}(\mathbf{y},\mathbf{y}')$ for all $\mathbf{y},\mathbf{y}'\in S$ belongs to the exponential rationals in a closed class of asymptotically monotone functions (CAM) [2, 3], the Markov chain is strongly ergodic.

Strongly ergodicity implies that the Markov chain has a unique stationary distribution π_T, where $\pi_T(\mathbf{y})$ is the probability of hitting point $\mathbf{y}$ during the search of CPSA.

Comparing the Δ_L for CSA and CPSA, although it depends on the user-defined neighborhood, the Δ_L for CPSA is usually smaller than the Δ_L for CSA. Since Δ_L is the maximum difference of the penalty function value between two neighboring points, CPSA has a smaller

Δ_L because its neighborhood is partitioned and two neighboring points only differ by a small subset of the variables, which leads to small differences in their penalty function values. In contrast, two neighboring points in CSA can differ in all variables and therefore have larger variations in their penalty function values. A smaller Δ_L for CPSA means that it can reduce the temperature and converge to CGM faster than CSA.

Asymptotic Convergence to Constrained Global Minima

In this section, we briefly overview the framework of generalized simulated annealing (GSA) [82] and show that the Markov chain modelling CPSA fits into this framework in which the Markov chain minimizes a virtual energy. Based on the results of GSA, we prove that CPSA has asymptotic convergence to a CGM_d.

Generalized Simulated Annealing (GSA). GSA [82] aims to establish a new framework for a class of stochastic search procedures broader than the SA algorithm, where the family of transition probabilities is given as follows:

> **Definition 3.1** *Communication Cost* "(Definition 2.1 in [82]). Let $(Q_T)_{T>0}$ be a family of Markov kernels on E. We say that $(Q_T)_{T>0}$ is admissible for q and k if there exists a family of positive real-valued numbers $(V(i,j))_{(i,j)\in E\times E}$ such that:
>
> - $V(i,j) < +\infty$, iff $q(i,j) > 0$,
> - for all $T > 0$, all $i, j \in E$,
>
> $$\frac{1}{\kappa} q(i,j) e^{-V(i,j)/T} \leq Q_T(i,j) \leq \kappa q(i,j) e^{-V(i,j)/T}, \tag{3.49}$$
>
> where function $V : E \times E \to [0, +\infty]$ is called the *communication cost function*."

Any one-step transition probability $Q_T(i,j)$ that satisfies the two conditions in Definition 3.1 is called generalized simulated annealing (GSA). In the following, we quote the notion of *A-graph* and *virual energy* as defined in [26, 82]:

> **Definition 3.2** *A-Graph* "(Definition 2.4 in [26, 82]). Let $A \subset E$. A set g of arrows $i \to j$ in $A^c \times E$ is an *A-graph*, where $j \in \mathcal{N}(i)$, iff a) for each $i \in A^c$, there exists a unique $j \in E$ such that $i \to j \in g$; b) for each $i \in A^c$, there is a path in g ending on a point $j \in A$."

Here E is the set of all states (or nodes) in the Markov chain defined in Definition 3.1, A is a subset of E, and A^c is the complement of A in E. Accordingly, A-graph g is actually a spanning tree rooted at set A, where the tree is defined over the digraph constructed by the neighborhood $\mathcal{N}(i)$. Let $G(A)$ be the set of A-graphs. For each $g \in G(A)$, the cost of g is defined as $V(g) = \sum_{i \to j \in g} V(i,j)$.

> **Definition 3.3** *Virtual Energy* "(Definition 2.5 in [82]). For each state $i \in E$, its *virtual energy* $W(i)$ is defined as:
>
> $$W(i) = \min_{g \in G(\{i\})} V(g), \tag{3.50}$$
>
> which is the cost of the minimum spanning tree rooted at point i."

The following theorem shows the asymptotic convergence of GSA in minimizing virtual energy $W(i)$.

Proposition 3.1 "(Proposition 2.6 in [26, 82]). For every $T > 0$, the unique stationary distribution π_T of the Markov chain satisfies:

$$\pi_T(i) \longrightarrow exp\left(-\frac{W(i) - W(E)}{T}\right) \quad \text{as} \quad T \longrightarrow 0, \tag{3.51}$$

where $W(i)$ is the virtual energy of i, and $W(E) = \min_{i \in S} W(i)$."

Asymptotic Convergence of CPSA. Our Markov chain (3.43) fits into the framework of GSA (3.49), if we define an irreducible Markov kernel $Q_T(i, j) = P_T(\mathbf{y}, \mathbf{y}')$ and its associated communication cost $V(\mathbf{y}, \mathbf{y}')$:

$$V(\mathbf{y}, \mathbf{y}') = \begin{cases} (L(\mathbf{y}') - L(\mathbf{y}))^+ & \text{if } \mathbf{y}' = (y', \alpha, \gamma) \\ \\ (L(\mathbf{y}) - L(\mathbf{y}'))^+ & \text{if } \mathbf{y}' = (y, \alpha', \gamma) \text{ or } \mathbf{y}' = (y, \alpha, \gamma'). \end{cases} \tag{3.52}$$

Obviously $V(\mathbf{y}, \mathbf{y}') \geq 0$ and function V: $S \times S \rightarrow [0, +\infty]$.

Since CPSA fits into the framework of GSA, according to the result of GSA, any procedure that can be modelled by GSA minimizes an implicit virtual energy $W(\mathbf{y})$, and converges to the global minimum of $W(\mathbf{y})$ with probability one. Here, virtual energy $W(\mathbf{y})$ is the cost of the minimum spanning tree rooted at point $\mathbf{y}$ of the digraph governed by $\mathcal{N}(\mathbf{y})$. Therefore, in order to prove that CPSA converges asymptotically to a CGM_d, we need to show that $W(\mathbf{y})$ is minimized at (y^*, α, γ) for certain α, γ. This is stated in the following theorem.

Theorem 3.8 The Markov chain modeling CPSA converges asymptotically to a CGM_d $y^* \in$

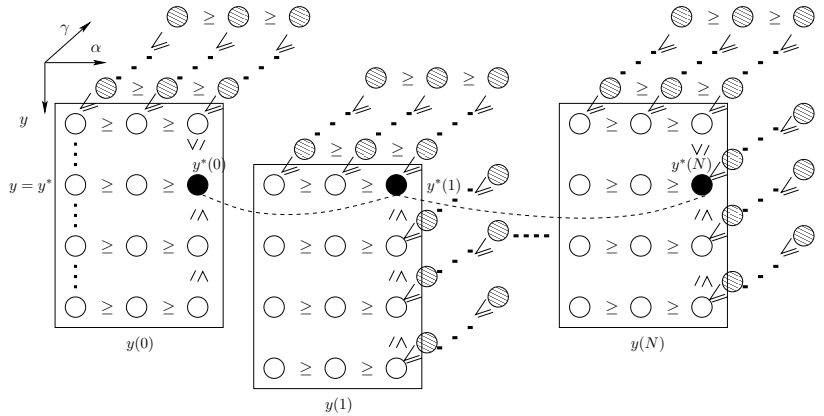


Figure 3.6: Proof strategy for Theorem 3.8.

Y_{opt} with probability one.

Proof. The proof consists of two steps as shown in Figure 3.6. First, we show that for a given y, the virtual energy satisfies $W((y, \alpha', \gamma)) \leq W((y, \alpha, \lambda))$ for any $\alpha' > \alpha$ and $W((y, \alpha, \gamma')) \leq W((y, \alpha, \lambda))$ for any $\gamma' > \gamma$. Second, we show that $W((y^*, \alpha_{max}, \gamma_{max})) < W((y, \alpha_{max}, \gamma_{max})$ at the maximum value of the penalty values, where $y^* \in Y_{opt}$ and $y \in \mathcal{D}^w - Y_{opt}$. Hence, $W(\mathbf{y})$ is minimized at $(y^*, \alpha_{max}, \gamma_{max})$, and the Markov chain converges to CGM_d $y^* \in Y_{opt}$ with probability one.

Figure 3.6 illustrates the key difference between the proof for CSA and CPSA in the second step. CSA was proved under an unpartitioned neighborhood, whereas CPSA is proved under a partitioned neighborhood here. We outline the proof strategy and point out the difference between the proof for CPSA and CSA before presenting the details. The proof is similar to the original proof for the asymptotic convergence of CSA [98]. The difference is in the second step, where we need to show that $W((y^*, \alpha_{max}, \gamma_{max})) < W((y, \alpha_{max}, \gamma_{max}))$

at the maximum value of the penalty values. Since the virtual energy $W((y^*, \alpha_{max}, \gamma_{max}))$ is defined to be the total cost of a minimum spanning tree rooted at y^*, the proof strategy in CSA is to construct a path from y to y^* in the minimum spanning tree of an arbitrary point $y \in \mathcal{D}^w - Y_{opt}$, reverse the path, and prove that we can construct a spanning tree of y^* that has less total weight than the minimum spanning tree of y. Since the minimum spanning tree of y^* has an even less total cost than the constructed tree, we can conclude that $W((y^*, \alpha_{max}, \gamma_{max})) < W((y, \alpha_{max}, \gamma_{max}))$.

The key difference, as illustrated in Figure 3.6, between the proof for CPSA and that for CSA is that, while the neighborhood of CSA is defined in the entire search space without partitioning, CPSA has partitioned neighborhoods. Therefore, the key point in our proof is to show that we can still construct a path from any point $y \in \mathcal{D}^w - Y_{opt}$ to a point $y^* \in Y_{opt}$ by linking the partitioned neighborhoods together, and that the spanning tree of y^* obtained from reversing the path has a total cost less than the cost of the minimum spanning tree rooted at y.

a) The proof for the first step is the same as that in the first step of the proof to Theorem 3.1 in Wang's Thesis on the asymptotic convergence of CSA [98], except that the penalty vectors (α, γ) here are equivalent to λ in Wang's Thesis.

b) From (a) , we have that, for any $y \in \mathcal{D}^w$, α, γ,

$$W((y, \alpha_{max}, \gamma)) \leq W((y, \alpha, \gamma)), \tag{3.53}$$

$$W((y, \alpha, \gamma_{max})) \leq W((y, \alpha, \gamma)), \tag{3.54}$$

which can be combined to get:

$$W((y, \alpha_{max}, \gamma_{max})) \leq W((y, \alpha, \gamma)), \tag{3.55}$$

for all $y \in \mathcal{D}^w$.

Accordingly, we only need to compare $W((y, \alpha_{max}, \gamma_{max}))$ (where $y \in \mathcal{D}^w - Y_{opt}$) with $W((y^*, \alpha_{max}, \gamma_{max}))$ (where $y^* \in Y_{opt}$) at the maximum $\alpha_{max}, \gamma_{max}$ of the penalty values.

Let $MT(\mathbf{y})$ be the minimum spanning tree of $\mathbf{y} = (y, \alpha_{max}, \gamma_{max})$ and its associated virtual energy be $W(\mathbf{y})$. There must exist a path $\mathcal{P} = \mathbf{y}_0 (= \mathbf{y}^*) \rightarrow \mathbf{y}_1 \rightarrow \cdots \rightarrow \mathbf{y}_{r-1} \rightarrow \mathbf{y}_r (= \mathbf{y})$ of length r from $\mathbf{y}^* = (y^*, \alpha_{max}, \gamma_{max})$ to $\mathbf{y}$ in $MT(\mathbf{y})$. This path can be constructed by linking all the partitioned neighborhoods together. In fact, we can find the following paths:

$$\mathbf{y} \in \mathcal{N}_0(\mathbf{y}_{0,1}),\ \mathbf{y}_{0,1} \in \mathcal{N}_0(\mathbf{y}_{0,2}),\ \cdots,\ \mathbf{y}_{0,i_0} \in \mathcal{N}_0(\mathbf{y}_0^*) \tag{3.56}$$

$$\mathbf{y}_\mathbf{0}^* \in \mathcal{N}_1(\mathbf{y}_{1,1}),\ \mathbf{y}_{1,1} \in \mathcal{N}_1(\mathbf{y}_{1,2}),\ \cdots,\ \mathbf{y}_{1,i_1} \in \mathcal{N}_1(\mathbf{y}_1^*) \tag{3.57}$$

$$\cdots$$

$$\mathbf{y}_{\mathbf{N-1}}^* \in \mathcal{N}_N(\mathbf{y}_{N,1}),\ \mathbf{y}_{N,1} \in \mathcal{N}_N(\mathbf{y}_{N,2}),\ \cdots,\ \mathbf{y}_{N,i_N} \in \mathcal{N}_N(\mathbf{y}_N^*) \tag{3.58}$$

where

$$\mathbf{y}_i^* = (y', \alpha_{max}, \gamma_{max}) \quad \text{and} \quad y'(j) = y^*(j), j = 0, \cdots, i.$$

Based on the definition of $\mathcal{N}_d(y(i))$, there is a path from $\mathbf{y}_{i-1}^*$ to $\mathbf{y}_i^*$ for $i = 1, \cdots, N$. This is true because the only different part between $\mathbf{y}_{i-1}^*$ and $\mathbf{y}_i^*$ is $y(i)$, and the connectivity requirement for the discrete neighborhood $\mathcal{N}_d(y(i))$ ensures that we can find a path from $y(i)$ to $y^*(i)$.

Based on (3.56) to (3.58), we have path from $\mathbf{y}$ to $\mathbf{y}^*$:

$$\mathbf{y} \rightarrow \mathbf{y}_{0,1} \rightarrow \mathbf{y}_{0,2} \cdots \rightarrow \mathbf{y}_0^* \rightarrow \mathbf{y}_{1,1} \rightarrow \mathbf{y}_{1,2} \cdots \rightarrow \mathbf{y}_1^*$$

$$\cdots \rightarrow \mathbf{y}_{N-1}^* \rightarrow \mathbf{y}_{N,1} \rightarrow \mathbf{y}_{N,2} \cdots \rightarrow \mathbf{y}_N^* = \mathbf{y}^*$$

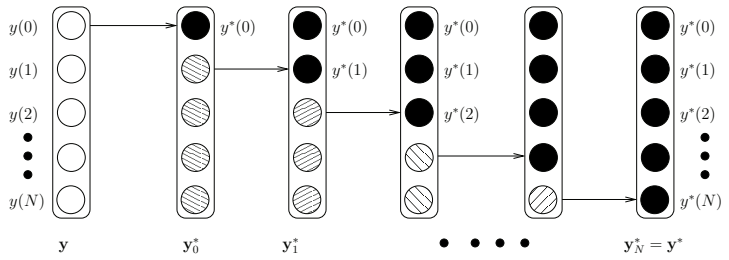


Figure 3.7: The construction of a path from $\mathbf{y}$ to $\mathbf{y}^*$.

Figure 3.7 illustrates the construction of a path from $\mathbf{y}$ to y^*. The white circles show the partitioned components $y(0)$ to $y(N)$ of $\mathbf{y}$, the black circles show $y^*(0)$ to $y^*(N)$ of $\mathbf{y}^*$, and the other circles are shaded because $y(t)$ may change during the transition due to variable sharing.

In the first step, we find a path from $\mathbf{y}$ to $\mathbf{y}_0^*$. Since the only different part between $\mathbf{y}$ and $\mathbf{y}_0^*$ is $y(0)$, we only need to find a path from $y(0)$ to $y^*(0)$. Such a path must exist due to the connectivity requirement for the discrete neighborhood $\mathcal{N}_d(y(0))$. After we transit from $y(0)$ to $y^*(0)$, the values of $y(1), \cdots, y(N)$ may also get changed to $y'(1), \cdots, y'(N)$ since they share some variables with $y(0)$.

In the second step, we find a path from $\mathbf{y}_0^*$ to $\mathbf{y}_1^*$. Since $y^*(0)$ is already reached, the only different part between $\mathbf{y}_0^*$ and $\mathbf{y}_1^*$ is $y(1)$ and we only need to find a path from $y'(1)$ to $y^*(1)$. Again, such a path must exist due to the connectivity requirement for the discrete neighborhood $\mathcal{N}_d(y(1))$. Therefore, we can continue this process until we reach $\mathbf{y}_N^* = (y^*(0), y^*(1), \cdots, y^*(N)) = \mathbf{y}^*$.

After reversing the path from $\mathbf{y}$ to $\mathbf{y}^*$, we obtain a path from $\mathbf{y}$ to $\mathbf{y}^*$ and also a spanning

tree $T(\mathbf{y}^*)$ of $\mathbf{y}^*$ with cost $C(\mathbf{y}^*)$, satisfying:

$$\begin{aligned} W(\mathbf{y}) - C(\mathbf{y}^*) &= \sum_{k=1}^{r} \{[L(\mathbf{y}_k) - L(\mathbf{y}_{k-1})]^+ - [L(\mathbf{y}_{k-1}) - L(\mathbf{y}_k)]^+\} \\ &= \sum_{k=1}^{r} [L(\mathbf{y}_k) - L(\mathbf{y}_{k-1})] = L(\mathbf{y}_r) - L(\mathbf{y}_0) \\ &= L(y, \alpha_{max}, \gamma_{max}) - L(y^*, \alpha_{max}, \gamma_{max}) > 0, \end{aligned}$$

based on the definition of γ. Because $W((y^*, \alpha_{max}, \gamma_{max})) \leq C((y, \alpha_{max}, \gamma_{max})$, we have $W((y^*, \alpha_{max}, \gamma_{max})) \leq C((y^*, \alpha_{max}, \gamma_{max})) < W((y, \alpha_{max}, \gamma_{max}))$.

c) Summarizing a) and b), we have that, for any $y \in D - Y_{opt}$, $y^* \in Y_{opt}$, α, and γ,

$$W((y^*, \alpha_{max}, \gamma_{max})) \leq W((y, \alpha, \gamma)). \tag{3.59}$$

Therefore, we conclude that virtual energy $W(\mathbf{y})$ of $\mathbf{y} = (y, \alpha, \gamma)$ is minimized at CGM_d $(y^*, \alpha_{max}, \gamma_{max})$. Thus, the Markov chain converges to CGM_d $y^* \in Y_{opt}$ with probability one according to Proposition 3.1.

3.4 Summary

In this section, we have developed a theory of extended saddle points to characterize the constrained local minimum of NLPs under constraint partitioning, proposed a search framework to look for points that satisfy the condition, and proved convergence results. Based on a penalty function, we have developed the necessary and sufficient ESPC conditions for constrained local minimum of discrete, continuous, and mixed NLPs. The conditions hold true over an extended region of penalty values that are larger than some thresholds.

Next, we have extended the ESPC conditions and have applied it to solve NLPs under constraint partitioning. After formulating the MINLP problem under constraint partitioning in a mathematical form, we have developed the concepts of neighborhood and constrained local minimum under constraint partitioning, and have derived a set of partitioned necessary conditions for resolving global constraints under constraint partitioning. The theory facilitates constraint partitioning by providing a set of necessary conditions, one for each subproblem, to characterize a constrained local minimum under constraint partitioning.

Finally, we have presented an iterative search framework for finding points that satisfy the partitioned ESPC condition. The framework searches in each subproblem by solving a modified problem with an objective function that is biased by global constraint violations, and performs ascents in the subspace of penalty values for violated global constraints. We have proved the asymptotic convergence of a partitioned search algorithm that performs stochastic descents and ascents using simulated annealing.

Chapter 4

Applications on Planning

A planning problem involves arranging actions and assigning resources in order to accomplish some 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 chapter, we apply constraint partitioning to solve planning problems. We study two planning models, including planning with the standard PDDL2.2 model and ASEPN planning for NASA's applications. For each planning application, we evaluate the constraint structure and locality of the planning problems by measuring the ratio of global constraints, and by identifying a suitable dimension for partitioning the constraints. We then develop two planning systems that partition the constraints of large planning problems, solve each subproblem using a basic planner, and study new strategies for resolving global inconsistencies. Finally we present experimental results to show that our proposed approach can improve significantly the solution time and quality over those of existing solvers.

4.1 Applications on PDDL2.2 Planning

In this section, we first discuss the constraint locality property in PDDL2.2 planning. We then present a subgoal partitioning strategy and its implementation in SGPlan_{g} for solving

temporal planning problems. The subscript "g" in SGPlan_g means that it partitions the constraints of a planning problem by its subgoal facts. Our strategy partitions the mutual-exclusion constraints of a temporal planning problem by its subgoals into subproblems, solves the subproblems individually, and resolves iteratively violated global constraints across the subproblems. We evaluate various heuristics for resolving global constraints and demonstrate the performance of SGPlan_g in solving the benchmarks of the Third and the Fourth International Planning Competitions.

4.1.1 Locality of mutual-exclusion constraints in temporal planning

In this section, we first define the mutual-exclusion constraints in a MINLP formulation of planning problems. Based on the structure of these constraints in IPC4 benchmarks, we show that partitioning by subgoals leads to constraints that can be localized better than partitioning by time.

Representation of Mutual-Exclusion Constraints

By following standard notations and definitions in the literature [37], we introduce in this section the basic definitions and the constraints used in a constrained formulation of planning problems.

Definition 4.1 A *planning problem* $\mathcal{T} = (\mathcal{O}, \mathcal{F}, \mathcal{I}, \mathcal{G})$ is a quadruple, where $\mathcal{O}$ is the set of possible actions in $\mathcal{T}$, $\mathcal{F}$ is the set of all facts, $\mathcal{I}$ is the set of initial facts, and $\mathcal{G}$ is the set of goal facts.

Definition 4.2 A *state* $S = \{f_1, \cdots, f_{n_S}\}$ is a subset of facts in $\mathcal{F}$ that are true.

Definition 4.3 A *temporal action* $o \in \mathcal{O}$ is associated with the following attributes:

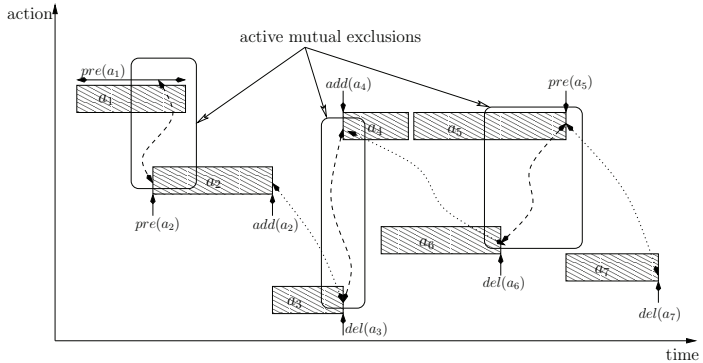


Figure 4.1: An example temporal plan. Active mutual exclusions between actions are shown as dashed lines, whereas inactive mutual exclusions are shown as dotted lines.

a) $pre(o)$, a set of facts that define the preconditions of action o;

b) $add(o)$, a set of facts that define the add-effects of o;

c) $del(o)$, a set of facts that define the delete-effects of o; and

d) $s(o)$ and $e(o)$, two quantities that define, respectively, the starting and the ending times of o.

The resulting state of applying action o to state S is defined as:

$$Result(S, o) = \begin{cases} (S \bigcup add(o)) \backslash del(o) & \text{if } pre(o) \in S \\ S & \text{if } pre(o) \notin S. \end{cases} \tag{4.1}$$

The resulting state of applying a sequence of actions to S is defined recursively as:

$$Result(S, (o_1, \cdots, o_n)) = Result(Result(S, (o_1, \cdots, o_{n-1})), o_n). \tag{4.2}$$

Definition 4.4 A *temporal plan* $\mathcal{P}_m = \{a_1, a_2, \cdots, a_m\}$ is a list of m temporal actions, where a_i has been assigned starting time $s(a_i)$ and ending time $e(a_i)$.

Figure 4.1 illustrates a temporal plan of seven actions. In each action, we indicate, where appropriate, its preconditions, add-effects, and delete-effects.

Concurrent actions in a plan must be arranged in such a way that observes mutual exclusions. The notion of *mutual exclusion* (mutex) was first proposed in GraphPlan [11]. It was defined for a planning graph, which is a level-by-level constraint graph that alternates between a fact level and an action level. The mutual-exclusion relations in a planning graph can be classified into transient (level-dependent) and persistent (level-independent) [11]. A mutual exclusion is *transient* if it exists only in certain levels of the graph and vanishes as more levels of the graph are built. In contrast, a mutual exclusion is *persistent* if it holds at every level until the fix-point level (the last level of the graph) is achieved. In this dissertation, we only consider level-independent, persistent mutual-exclusion relationships, as transient mutual exclusions are used exclusively for searches in GraphPlan.

Actions a and b are marked as *persistently mutual exclusive* when one of the following occurs.

a) Actions a and b have persistent *competing needs*,[1] where competing needs are represented by the persistent mutual exclusion of the preconditions of a and those of b;

b) Actions a and b have persistent *inconsistent effects*, when one of the actions deletes an add-effect of the other.

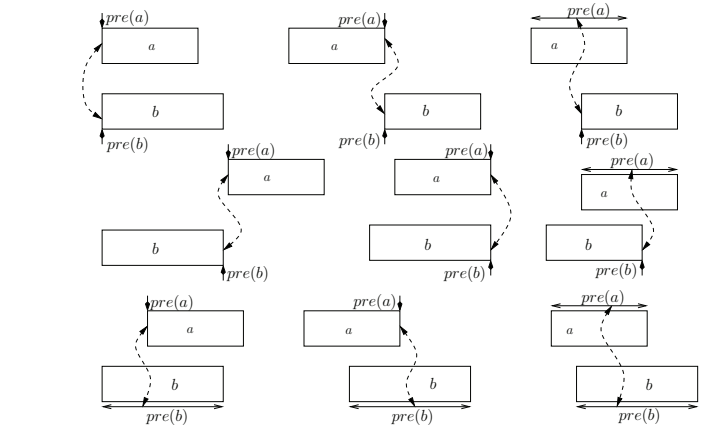
c) Actions a and b have persistent *interference*, when one of the actions deletes a precondition of the other.

[1]The terms "competing needs," "inconsistent effects," and "interference" were originally proposed for GraphPlan [11].

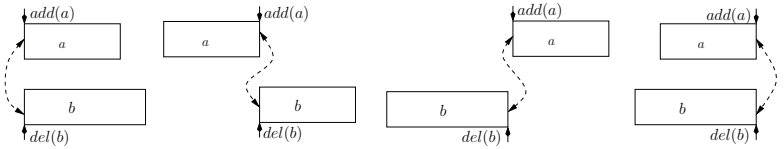
Two facts p and q are *persistently mutual exclusive* if all possible ways of making p true are persistently exclusive with all possible ways of making q true; that is, each action a having p as an add-effect ($p \in add(a)$) is persistently mutual exclusive with each action b having q as an add-effect ($q \in add(b)$). For simplicity, in the rest of this dissertation, we use the terms mutual-exclusion actions and facts to refer to the corresponding persistent mutual-exclusion actions and facts.

Given a temporal plan, a mutual-exclusion relationship can be *active* or *inactive*. Figure 4.2 illustrates the possible scenarios when a mutual-exclusion relationship is active. Note that for temporal actions, a precondition fact can be effective at the beginning, at the end, or during the entire duration of an action; whereas an add or delete effect can be effective only at the beginning or at the end of an action.

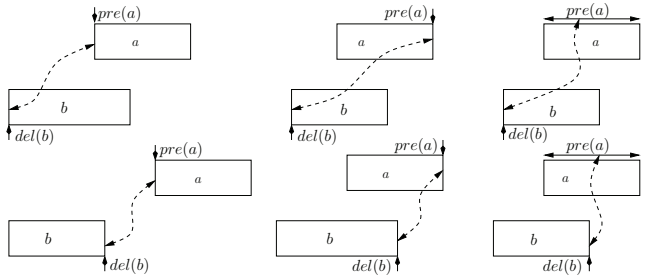
When two actions a and b are mutual exclusive due to competing needs, that is, when a precondition fact p_a of action a is mutual exclusive with a precondition fact p_b of action b, the mutual-exclusion relationship between a and b is active in a plan when one of the



a) Nine scenarios to activate a mutual-exclusion relationship between two actions a and b with competing needs. ($pre(a)$ is assumed to have competing needs to $pre(b) : \exists f_1 \in pre(a), f_2 \in pre(b), f_1$ and f_2 are mutual exclusive)



b) Four scenarios to activate a mutual-exclusion relationship between two actions a and b with inconsistent effects. ($add(a)$ is assumed to be overlapping with $del(b)$: $add(a) \cap del(b) \neq \emptyset$)



c) Six scenarios to activate a mutual-exclusion relationship between two actions a and b with interference. ($pre(a)$ is assumed to be unsupported and overlapping with $del(b)$: $pre(a) \cap del(b) \neq \emptyset$)

Figure 4.2: Various scenarios in the activation of mutual-exclusion relationships between actions a and b. A solid arrow shows the case when a precondition, an add effect, or a delete effect is effective at the beginning, the end, or the entire duration of an action. An active mutual exclusion is shown as a dashed arrow.

following nine conditions is satisfied (see Figure 4.2a):

$$\begin{cases} s(a) = s(b) & \text{if } p_a \text{ is ``AT START'' and } p_b \text{ is ``AT START''} \\ e(a) = s(b) & \text{if } p_a \text{ is ``AT END'' and } p_b \text{ is ``AT START''} \\ s(a) \leq s(b) \leq e(a) & \text{if } p_a \text{ is ``OVERALL'' and } p_b \text{ is ``AT START''} \\ s(a) = e(b) & \text{if } p_a \text{ is ``AT START'' and } p_b \text{ is ``AT END''} \\ e(a) = e(b) & \text{if } p_a \text{ is ``AT END'' and } p_b \text{ is ``AT END''} \\ s(a) \leq e(b) \leq e(a) & \text{if } p_a \text{ is ``OVERALL'' and } p_b \text{ is ``AT END''} \\ s(b) \leq s(a) \leq e(b) & \text{if } p_a \text{ is ``AT START'' and } p_b \text{ is ``OVERALL''} \\ s(b) \leq e(a) \leq e(b) & \text{if } p_a \text{ is ``AT END'' and } p_b \text{ is ``OVERALL''} \\ s(b) \leq e(a) \text{ and } s(a) \leq e(b) & \text{if } p_a \text{ is ``OVERALL'' and } p_b \text{ is ``OVERALL.''} \end{cases} \tag{4.3}$$

When actions a and b are mutual exclusive due to inconsistent effects, that is, when an add effect a_a of action a is also a delete effect d_b of action b, the mutual-exclusion relationship between a and b is active in a plan when one of the following four conditions is satisfied (Figure 4.2b):

$$\begin{cases} s(a) = s(b) & \text{if } a_a \text{ is ``AT START'' and } d_b \text{ is ``AT START''} \\ e(a) = s(b) & \text{if } a_a \text{ is ``AT END'' and } d_b \text{ is ``AT START''} \\ s(a) = e(b) & \text{if } a_a \text{ is ``AT START'' and } d_b \text{ is ``AT END''} \\ e(a) = e(b) & \text{if } a_a \text{ is ``AT END'' and } d_b \text{ is ``AT END.''} \end{cases} \tag{4.4}$$

When actions a and b are mutual exclusive due to interference, that is, when an unsupported precondition effect p_a of action a is also a delete effect d_b of action b, the mutual

exclusion between a and b is active in a plan when one of the following six conditions is satisfied (Figure 4.2c):

$$\begin{cases} s(a) \geq s(b) & \text{if } p_a \text{ is "AT START" and } d_b \text{ is "AT START"} \\ e(a) \geq s(b) & \text{if } p_a \text{ is "AT END" and } d_b \text{ is "AT START"} \\ e(a) \geq s(b) & \text{if } p_a \text{ is "OVERALL" and } d_b \text{ is "AT START"} \\ s(a) \geq e(b) & \text{if } p_a \text{ is "AT START" and } d_b \text{ is "AT END"} \\ e(a) \geq e(b) & \text{if } p_a \text{ is "AT END" and } d_b \text{ is "AT END"} \\ e(a) \geq e(b) & \text{if } p_a \text{ is "OVERALL" and } d_b \text{ is "AT END."} \end{cases} \tag{4.5}$$

Note that in the last case, actions a and b do not have to overlap in their durations in order to activate the mutual exclusion. However, in the first two cases, actions a and b must overlap or be adjacent to each other in order to activate the mutual exclusion.

Definition 4.5 A temporal plan $\mathcal{P}$ is *conflict-free* if there exists no pair of actions a and b in $\mathcal{P}$ such that a and b have an active mutual-exclusion relationship.

The example plan in Figure 4.1 is not conflict-free, as there exist pairs of active mutual-exclusion actions. Actions a_1 and a_2 have active mutual exclusions due to competing needs; a_3 and a_4 have active mutual exclusions due to inconsistent effects; and a_5 and a_6 have active mutual exclusions due to interference. Other mutual exclusions (between a_2 and a_3, between a_4 and a_6, and between a_5 and a_7) are not active because they do not satisfy the conditions in (4.3)-(4.5).

Definition 4.6 A *solution plan* $\mathcal{P}$ to a temporal planning problem $\mathcal{T} = (\mathcal{O}, \mathcal{F}, \mathcal{I}, \mathcal{G})$ is a plan that is conflict-free, and that all facts $f \in \mathcal{G}$ are true in the resulting state $S = Result(\mathcal{I}, \mathcal{P})$

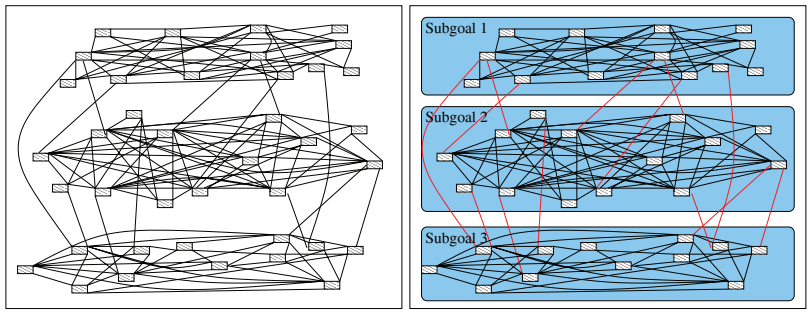
of applying the actions in $\mathcal{P}$ according to their starting times to $\mathcal{I}$. That is, $f \in S$ for all $f \in \mathcal{G}$.

Locality of Mutual-Exclusion Constraints in IPC4 Benchmarks

In this section, we evaluate the partitioning of mutual-exclusion constraints for IPC4 benchmarks. Our analysis shows the strong locality of these constraints when they are partitioned by problem subgoals as compared to partitioned by time. We do not study other partitioning criteria in this paper because they may lead to subproblems whose initial states and subgoals are not specified. Such subproblems will be hard to solve by existing planners because they may require a systematic enumeration of their initial states and subgoals in finding feasible plans.

Figure 4.3a shows the 93 mutual-exclusion constraints in a solution plan to the 4^{th} instance of the IPC4 Airport-Temporal domain. The instance involves moving three planes in an airport to designated gates. Each rectangular box in the figure represents an action, whereas a line joining two actions represents a mutual-exclusion constraint (that may be inactive). Figure 4.3b shows the partitioning of the actions into three subproblems, each involving the movement of one plane. We show local constraints (those that are relevant to actions in one subproblem) in solid lines and global constraints relating actions in different subproblems in dashed lines. It is clear that a majority of the constraints (84%) are local after partitioning them by subgoals.

To demonstrate the localization of mutual-exclusion constraints after partitioning them by subgoals, we analyze all the IPC4 instances by a modified Metric-FF planner [37] that supports the new features in PDDL2.2, such as temporal actions and derived predicates. For each instance, we use the modified Metric-FF planner to find an initial subplan for each of the subgoals. We then find all the mutual exclusions among the actions, including active



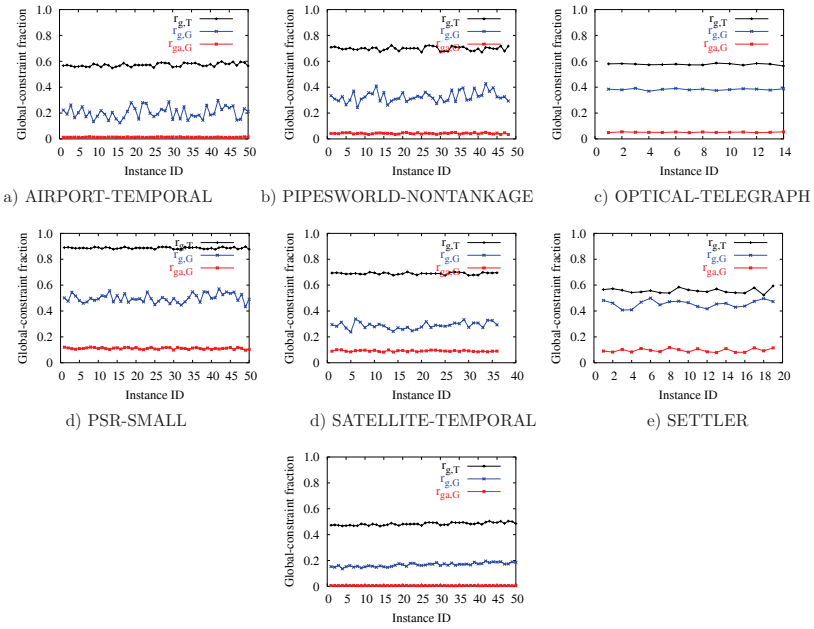
a) 93 mutual-exclusion constraints among actions b) 14 global constraints after partitioning

Figure 4.3: Mutual-exclusion constraints in the 4^{th} instance of the IPC4 AIRPORT-TEMPORAL variant in the Airport domain. Each rectangular box represents an action, whereas a line joining two actions represents a mutual-exclusion constraint (that may be inactive). Most constraints (79 out of 93 or 84%) are local constraints after partitioning them by subgoals. Global constraints are shown in dashed lines in (b).

and inactive ones. Finally, we compute the number of global constraints relating actions in different subplans, as well as the number of initial active global constraints based on the subplan evaluated for each subgoal.

As a comparison, we also evaluate the partitioning of the constraints of each instance by its temporal horizon, which we proposed in our previous work [83, 15]. In each instance, we use the modified Metric-FF planner to find an initial plan, set the number of temporal stages to be the same as the number of subgoals, and partition the horizon of the solution plan evenly into multiple stages. We then count the number of local constraints in each stage and the number of global constraints relating actions in different stages.

Figure 4.4 illustrates the results for seven IPC4 domains. For each instance, let N_c be the total number of mutual-exclusion constraints, N_g^T be the number of global constraints under constraint partitioning by time, N_g^G be the number of global constraints under constraint partitioning by subgoals, and N_{ga}^G be the number of initial active global constraints under



a) AIRPORT-TEMPORAL b) PIPESWORLD-NONTANKAGE c) OPTICAL-TELEGRAPH

d) PSR-SMALL d) SATELLITE-TEMPORAL e) SETTLER

f) UMTS-TEMPORAL

Figure 4.4: Variations of $r_{g,T}$, $r_{g,G}$, and $r_{ga,G}$ across the instances of seven IPC4 domains.

constraint partitioning by subgoals. We then compute the following ratios:

$r_{g,T} = (N_g^T/N_c)$: fraction of global constraints under constraint partitioning by time,

$r_{g,G} = (N_g^G/N_c)$: fraction of global constraints under constraint partitioning by subgoals,

$r_{ga,G} = (N_{ga}^G/N_c)$: fraction of initial active global constraints under subgoal partitioning.

The results show that constraint partitioning by subgoals leads to a lower fraction of global constraints than that of constraint partitioning by time, and that the fractions do not vary

significantly across the instances of a domain. The results also show that the fraction of initial active global constraints under subgoal partitioning is very small. Except for two domains (PSR-Small and Settler), the fractions of initial active global constraints are consistently less than 0.1. This behavior is important because only active global constraints will need to be resolved during planning, and the number of such constraints should decrease as planning progresses. We describe in Section 4.1.3 two strategies for reducing the number active global constraints in planning.

Since the fractions of global constraints in a domain do not vary significantly across different instances, we summarize for each domain the average fraction of global constraints under constraint partitioning by time and that under constraint partitioning by subgoals, as well as the average fraction of initial active global constraints under constraint partitioning by subgoals. The results show that constraint partitioning by subgoals consistently lead to a lower fraction of global constraints than constraint partitioning by time, and that the fraction of initial active global constraints under constraint partitioning by subgoals is very small.

4.1.2 System architecture of SGPlan$_g$

Figure 4.5 shows the design of SGPlan$_g$ that implements the partition-and-resolve procedure. At the global level, SGPlan$_g$ partitions a planning problem into N subproblems, $G_1, \cdots, G_N$, one for each subgoal. It then orders the subproblems, evaluates the composed plans, and updates the penalties of violated global constraints. At the subgoal level, it applies landmark analysis to further partition G_i into c_i subproblems $P_{i,1}, \cdots, P_{i,c_i}$. For each subproblem, it performs subspace-reduction analysis to prune irrelevant facts and actions, and calls a modified Metric-FF planner to find a feasible local plan that reaches the local subgoal and that minimizes the penalty function.

Table 4.1: Average $r_{g,T}$, $r_{g,G}$, $r_{ga,G}$ across the instances of IPC4 domains. Boxed numbers are less than 0.1.

Domain Variant	$\overline{r}_{g,T}$	$\overline{r}_{g,G}$	$\overline{r}_{ga,G}$	Domain Variant	$\overline{r}_{g,T}$	$\overline{r}_{g,G}$	$\overline{r}_{ga,G}$
AIRPORT-STRIPS	0.557	0.219	0.017	AIRPORT-TEMPORAL	0.568	0.208	0.014
AIRPORT-TIMEWINDOWS	0.494	0.184	0.013	PIPES-NONTANKAGE	0.695	0.313	0.044
PIPES-TANKAGE	0.687	0.677	0.070	PIPES-DEADLINES	0.674	0.297	0.033
PIPES-DEADLINES-CO	0.682	0.296	0.039	OPTICAL-TELEGRAPH	0.575	0.399	0.052
OPTICAL-TELEGRAPH-DP	0.759	0.265	0.020	OPTICAL-TELEGRAPH-FL	0.799	0.426	0.037
PHILOSOPHER-DP	0.855	0.576	0.019	PHILOSOPHER-FL	0.822	0.507	0.087
PHILOSOPHER	0.554	0.370	0.066	PSR-MIDDLE	0.896	0.504	0.092
PSR-MIDDLE-COMPILED	0.882	0.478	0.049	PSR-LARGE	0.902	0.665	0.096
PSR-SMALL	0.897	0.489	0.114	SATELLITE-STRIPS	0.689	0.288	0.096
SATELLITE-NUMERIC	0.288	0.305	0.078	SETTLERS	0.549	0.451	0.100
SATELLITE-COMPLEX	0.642	0.282	0.069	SATELLITE-COMP-TIL	0.633	0.124	0.041
SATELLITE-TIME	0.686	0.289	0.093	SATELLITE-TIME-TIL	0.648	0.114	0.027
SATELLITE-COMP-TIL-CO	0.698	0.153	0.042	SATELLITE-TIL-CO	0.633	0.307	0.075
UMTS-TEMPORAL	0.463	0.157	0.006	UMTS-FLAW	0.459	0.136	0.006
UMTS-TEMPORAL-TIL	0.437	0.126	0.008	UMTS-FLAW-TIL	0.428	0.110	0.008
UMTS-TEMPORAL-TIL-COMP	0.407	0.098	0.008	UMTS-FLAW-TIL-COMP	0.414	0.086	0.007

The partition-and-resolve approach in SGPlan$_g$ is different from incremental planning [50] that uses a goal agenda. In incremental planning, a planner maintains a set of target facts, adds goal states incrementally into the target set, and extends the solution by using the new target set. This means that a goal state will always be satisfied once it is satisfied in an earlier target set. Because the search space increases as more goal states are added in the target set, it may be more expensive to solve subsequent problems with larger target sets. Moreover, it is difficult to tell which goals should be satisfied before others. In contrast, SGPlan$_g$ always addresses one goal fact at a time in a subproblem, without increasing its search space as other subproblems are solved.

Global-level techniques

Resolution of violated global constraints. We apply the partition-and-resolve procedure in Figure 5.2 to resolve violated global constraints in SGPlan$_g$. Our procedure alternates between finding feasible plans for all subgoals and updating the penalties of violated global

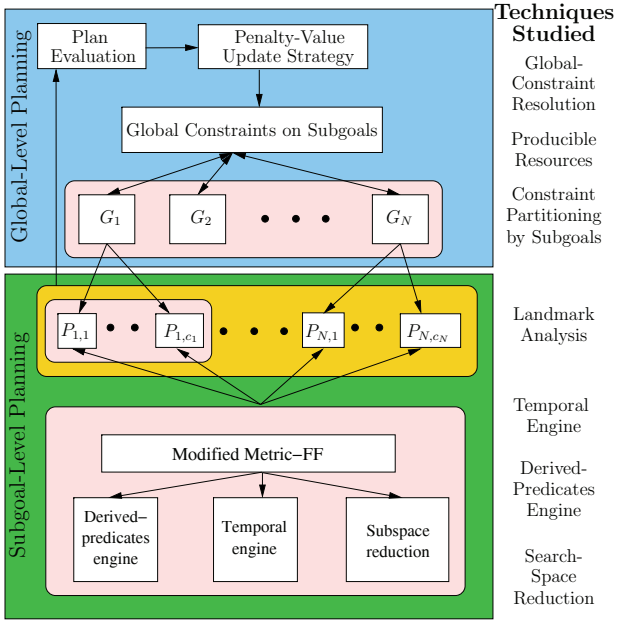


Figure 4.5: SGPlan$_g$: A planner implementing the partition-and-resolve procedure in Figure 5.2.

1. **procedure SGPlan$_g$**
   /* global-level planning */
2. parse problem file and preprocess;
3. **repeat** /* subgoal-level planning */
4. **for** each goal fact in the subgoal list
5. perform landmark analysis to generate a list of subproblems;
6. **for** each subproblem in the list
7. call search space-reduction procedure to eliminate irrelevant actions;
8. call basic planner (modified Metric-FF) to solve the subproblem;
9. **end_for**
10. **end_for**
11. evaluate plan z and update penalty values of violated global constraints;
12. periodically re-order the subgoals;
13. **until** feasible solution plan has been found or time limit is exceeded;
14. **end_procedure**

Figure 4.6: SGPlan$_g$: A planner implementing the partition-and-resolve procedure in Figure 5.2.

constraints.

For subgoal g_i, $i = 1, \ldots, N$, we use a basic planner to find a feasible plan z_i, while trying to minimize the local ℓ_1-penalty function defined as follows:

$$\Gamma(z_i) = J(z) + \sum_{\substack{j=1,\cdots,N,\\ j\neq i}} \gamma_{i,j} \times m_{i,j}, \tag{4.6}$$

where $m_{i,j}$ is the number of active global constraints between the actions in the subplan for g_i and those for g_j. To limit the number of penalties while characterizing the priorities among the subgoals, we assign a single penalty $\gamma_{i,j}$ for each pair of subgoals g_i and g_j, instead of a penalty for each global constraint between g_i and g_j. In SGPlan$_g$, we have adopted an implementation in LPG1.2 [30] for detecting persistent mutual exclusions and have implemented the conditions in (4.3)-(4.5) to determine if a mutual exclusion is active or not.

After solving all the subproblems, we evaluate the composed plan, find all the active global constraints, and update their penalty values in order to bias the search in the next iteration towards resolving them. In Section 4.1.3, we evaluate two strategies for updating penalty values and compare their effectiveness.

Producible resources. In some planning problems, there may be facts that can be made true anytime when needed and numerical resources that can be produced anytime we necessary. For example, in the Settlers domain, coal can always be produced in a mine. We define these producible logical and numerical resources as follows:

a) A fact is producible if it is an add-effect of an action with zero precondition, or an action whose preconditions are all producible.

b) A numerical resource is producible if it is increased by an action with zero precondition,

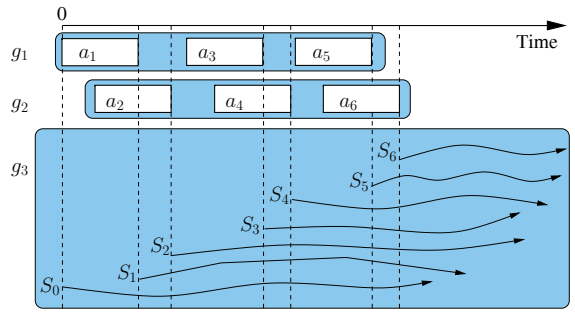


Figure 4.7: Generating multiple starting states for Subgoal g_3. S_0 is the initial state, whereas $S_i, i = 1, \ldots, 6$, is the state at the time action a_i is finished. SGPlan$_g$ generates a local subplan from each starting state and evaluates (4.6) in order to find a subplan that improves the penalty function.

or an action whose preconditions are all producible.

The planning tasks will be significantly easier if producible facts and resources can be detected in the preprocessing phase and be made available during planning. In SGPlan$_g$, we first identify all these facts and resources and derive a relaxed initial state by setting all producible facts to be true and all producible numerical resources to be large enough. After finding a feasible plan from the relaxed initial state, SGPlan$_g$ removes the unused numerical resources in the initial state and plans again. The process is repeated until there are no redundant initial resources. At that point, SGPlan$_g$ inserts into the beginning of the plan the necessary actions to generate the minimum initial producible resources needed.

Subgoal-level techniques

Evaluating multiple subplans for a subgoal. In finding a local feasible subplan for a subgoal that improves (4.6), SGPlan$_g$ generates a number of subplans from multiple starting states. Since no active global constraints exist between two identical subplans, we generate multiple starting states for a given subgoal by applying all possible prefix actions from each

of the other subgoals. For example, given the six actions planned in g_1 and g_2, there are six possible starting states when developing a subplan for g_3. For each starting state, SGPlan$_g$ generates a local feasible subplan by gradient descents from the starting state and accepts the subplan if it improves (4.6). If no better subplans can be found from all possible starting states, SGPlan$_g$ leaves the local subplan unchanged and moves on to the next subgoal.

Landmark analysis. First proposed by Porteous, Sebastia, and Hoffmann [66], landmark analysis allows a large planning problem to be decomposed into a series of much simpler subproblems. It aims to find some intermediate facts that must be true in any feasible plan, given the initial and the goal states. For example, assume that object O is to be delivered from A to D, and that the only path from A to D is $A \rightarrow B \rightarrow C \rightarrow D$. Then facts $AT(O, B)$ and $AT(O, C)$ are both landmark facts, since any feasible plan must make them true before reaching goal state $AT(O, D)$.

Originally, landmark analysis was used to find intermediate facts to reach all the subgoals of a planning problem [66]. In SGPlan$_g$, we first partition the constraints of a problem by their subgoals, before applying landmark analysis to find intermediate facts to reach each subgoal individually.

For each subgoal, we find landmarks using a relaxed planning approach. Given planning subproblem $\mathcal{T} = (\mathcal{O}, \mathcal{F}, \mathcal{I}, \mathcal{G})$, we first test for each fact $f \in \mathcal{F}$ to see if it is a landmark fact. We construct from initial state $\mathcal{I}$ a relaxed planning graph by ignoring the delete effects, and by enforcing f to be false at each level (even if it is made true by some actions). As a result, all the actions preconditioned by f will not be included in the relaxed planning graph. After the planning graph has been constructed, we check whether all the goal facts in goal state $\mathcal{G}$ are achieved. We know that f is a landmark fact and must be reached in any plan for the relaxed problem if there exists a goal fact that cannot be reached. Further, any feasible plan for the original problem must reach f at least once. This is true because any solution plan

of the original problem is also a solution plan of the relaxed problem.

After finding all the landmark facts of a subgoal, we build a sequential list of subproblems joined by landmark facts, and apply the basic planner to achieve each subproblem in order.

Search-space reduction. After partitioning a planning problem into subproblems, we can often eliminate many irrelevant actions in the search space of each subproblem before solving it. Such reductions are not very useful in planning problems that are not partitioned because all their actions are generally relevant.

As an example, consider a transportation domain whose goal is to move packages, drivers, and trucks to various locations, given an initial configuration. Suppose in a problem instance, the goal set is {(AT D1 S1), (AT T1 S1), (AT P1 S0), (AT P2 S0)}, given two packages P1 and P2, one driver D1, one truck T1, and two locations S1 and S2. Without partitioning, all the actions are relevant in resolving the subgoals. On the other hand, after partitioning, the actions for moving P2 around are irrelevant in the subproblem of resolving (AT P1 S0) and can be eliminated. Similarly, those actions for moving P1 or P2 are irrelevant in the subproblem of resolving (AT D1 S1).

We have designed a *backward relevance analysis* to eliminate some irrelevant actions in a subproblem before solving it by the basic planner. In the analysis, we maintain an *open list* of unsupported facts, a *close list* of relevant facts, and a *relevance list* of relevant actions. In the beginning, the open list contains only the subgoal facts of the subproblem, 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 and exclude those irrelevant actions. This analysis takes polynomial time.

Note that our reduction analysis is not complete, since the relevance list may still contain some irrelevant actions. For example, we can further reduce the relevance list by a forward analysis and by finding all applicable actions from the initial states before the backward analysis. However, further analysis may not be cost effective for reducing the overhead in planning.

Modified Metric-FF basic planner. After landmark analysis, each subproblem associated with a subgoal may be decomposed further into subproblems bounded by landmark facts. SGPlan_{g} solves each subproblem identified (or the original subgoal in case no landmark facts are identified) using a modified Metric-FF planner [37]. We have made several modifications in Metric-FF in order to entertain the new features in PDDL2.2.

The original Metric-FF can only solve problems in PDDL2.1 with propositional actions but does not support any temporal features. We have extended the parser of Metric-FF to support the full PDDL2.2 syntax and the definition of actions from atomic logical to durational temporal. The planning process has also been extended from sequential propositional planning to parallel temporal planning. Specifically, in the original Metric-FF, actions with an atomic length of one unit are ordered sequentially. In our modified Metric-FF planner, actions can be arranged in parallel, each with a numerical duration pre-defined in the problem specification.

We have also extended Metric-FF to support a new feature called derived predicates introduced in PDDL2.2. Derived predicates define axioms whose derived facts are derived by a set of precondition facts. For example, in a domain with boxes, if box A is above B and B is above C, then a derived predicate that A is above C can be generated: if ($above(A, B)$ and $above(B, C)$) then $above(A, C)$. Derived predicates can only appear in preconditions and goals but not in effects.

In our modified Metric-FF, we have implemented a technique proposed in MIPS 2.2 [22]

for handling derived predicates. We encode any derived predicate d as a special action o, where the precondition facts of o are the preconditions facts of d, the add-effects of o are the derived facts of d, and the delete-effect of o is empty. During the heuristic-function computation of Metric-FF, all these "derived-predicate actions" are also included in the relaxed plan. However, the heuristic value only counts the number of real actions in the relaxed plan but not the number of "derived-predicate actions." Also, only real actions are considered as candidates for forward expansion in any state. At any state, we expand the set of true facts by applying all applicable derived predicates iteratively until we reach a fixed-point state where no more true facts can be added.

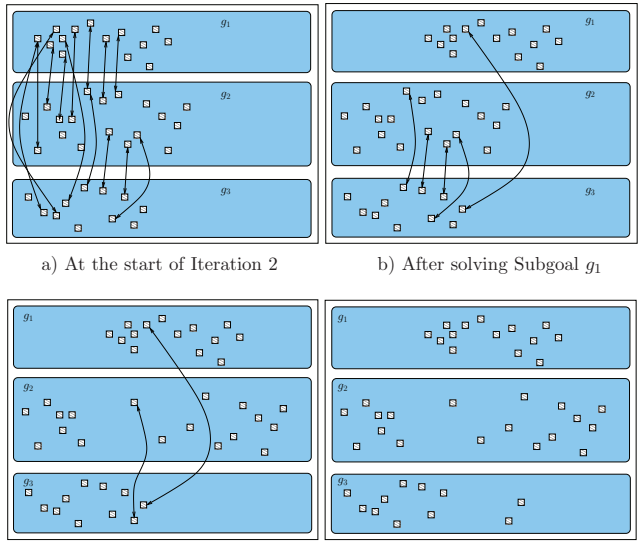
4.1.3 Updating the penalties of violated global constraints

In this section, we evaluate two strategies for updating the penalties of violated global constraints and demonstrate that SGPlan$_g$ can resolve them effectively.

SGPlan$_g$ first initializes the penalties of all global constraints when it starts. In the first iteration, SGPlan$_g$ solves each subgoal individually, without considering their global constraints. It then combines all the subplans into an integrated plan in order to determine the initial active global constraints across the subgoals. In subsequent iterations, SGPlan$_g$ finds a local feasible plan for each subgoal, while minimizing the weighted sum of violated global constraints. At the end of each iteration, SGPlan$_g$ increases the penalty of a violated global constraint in proportion to its violation. The process ends when all the constraints are satisfied.

We have designed two strategies for initializing the penalty of global constraints. The first strategy S_A sets a very large initial value for all penalties and updates them by rate α:

$$(S_A) \qquad \gamma_{i,j}^{(0)} = \xi, \quad \gamma_{i,j}^{(k)} = \gamma_{i,j}^{(k-1)} + \alpha \times m_{i,j}, \qquad k = 1, 2, \ldots, \tag{4.7}$$



a) At the start of Iteration 2 b) After solving Subgoal g_1

c) After solving Subgoal g_2 d) After solving Subgoal g_3

Figure 4.8: The planning process of SGPlan$_g$ using S_A in the second iteration in solving the AIRPORT-TEMPORAL-14 instance. Each box corresponds to an action in a subplan, whereas each arrow corresponds to an active global constraint.

where $\gamma_{i,j}^{(k)}$ is the penalty for the global constraints between g_i and g_j in the k^{th} iteration, $m_{i,j}$ is the number of active global constraints between g_i and g_j defined in (4.6), ξ is a large initial value, and α is a parameter controlling the rate of penalty updates. We set $\xi = 100$ and $\alpha = 0.1$ in our experiments. Intuitively, S_A tries to minimize the global-constraint violations at the beginning of the planning process. As a result, the subplans will have lower concurrency, and the number of global constraints will be reduced quickly. The strategy is effective for solving most of the IPC4 instances and was employed in SGPlan$_g$ in the IPC4 competition.

Figure 4.8 illustrates the planning process of SGPlan$_g$ using S_A on the AIRPORT-

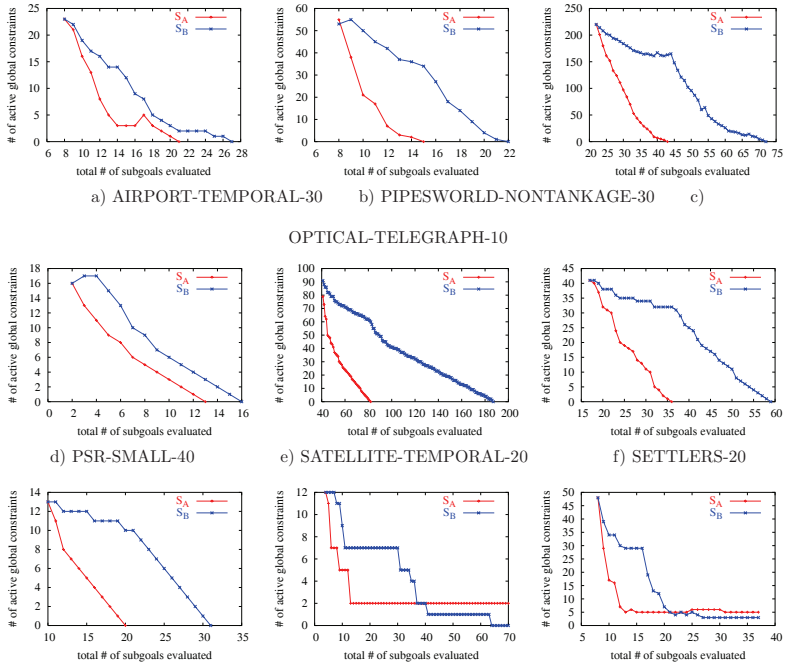
TEMPORAL-14 instance. Given the three subgoals in this instance, SGPlan$_g$ first evaluates each subgoal once in the first iteration in order to determine the initial active global constraints. The figure shows, respectively, the subplans and the active global constraints after evaluating the first, second, and third subgoal in the second iteration. The strategy is effective and reduces the number of active global constraints quickly from 14 in the beginning to zero in just one iteration.

The first strategy, however, does not work well for some domains, such as PIPESWORLD-DEADLINE, where temporal concurrency has to be exploited heavily in order to meet temporal deadlines. It has difficulty in satisfying deadlines in individual subgoals because it places too much emphasis on satisfying violated global constraints. To address this issue, the second strategy S_B sets the initial penalty values to zero and gradually increases them in each iteration:

$$(S_B) \qquad \gamma_{i,j}^{(0)} = 0, \quad \gamma_{i,j}^{(k)} = \gamma_{i,j}^{(k-1)} + \alpha \times m_{i,j}, \qquad k = 1, 2, \ldots \tag{4.8}$$

Figure 4.9 illustrates the performance of S_A and S_B on nine representative instances of the IPC4 domains. For each instance, we plot the progress of SGPlan$_g$ by showing the remaining number of active violated global constraints with respect to the total number of subgoals evaluated, where a subgoal evaluation involves the execution of Lines 5-9 in Figure 4.6. We use the number of subgoals evaluated, instead of the number of iterations through all the subgoals, because the number of active global constraints changes after evaluating each subgoal.

Figure 4.9 shows that active violated global constraints can be resolved efficiently by SGPlan$_g$ in most cases. We observe that S_A is better than S_B in most of the instances, and that the remaining number of active violated global constraints is generally reduced in an almost linear fashion with respect to the number of subgoals evaluated.



a) AIRPORT-TEMPORAL-30 b) PIPESWORLD-NONTANKAGE-30 c) OPTICAL-TELEGRAPH-10

d) PSR-SMALL-40 e) SATELLITE-TEMPORAL-20 f) SETTLERS-20

g) UMTS-TEMPORAL-50 h) PIPESWORLD-DEADLINE-05 i) PIPESWORLD-DEADLINE-10

Figure 4.9: Resolution of active violated global constraints in nine IPC4 instances using S_A and S_B for updating penalty values. In each instance, we plot the remaining number of active global constraints with respect to the total number of subgoals evaluated during planning. The x axis includes the number of subgoals evaluated in the first iteration in order to determine the active global constraints.

Both S_A and S_B, however, has difficulty in solving the PIPESWORLD-DEADLINE-10 instance (Figure 4.9i). For this domain, SGPlan$_g$ can only solve two instances (1 and 2) using S_A and eight instances (1, 2, 5, 6, 8, 14, 22, and 30) using S_B (Figure 4.9h). Although the number of initial active violated global constraints is small for this domain (an average of 3.3% of all constraints as shown in Table 4.1), both strategies may get stuck at some infeasible solutions and cannot make progress afterwards. The reason is that both strategies

do not have enough backtracking to generate new candidate subplans for each subgoal. As a result, both keep generating the same subplan at some point, regardless of how the violated constraints are penalized.

One way to improve convergence is to use backtracking within each run of Metric-FF in order to generate more candidate plans and to find new descent direction for the penalty function. Currently, our modified Metric-FF returns when it finds the first feasible plan. Another way is to reduce the number of subproblems, which will lead to larger subproblems with less global constraints to be resolved.

4.1.4 Experimental results

In this section, we compare the performance of SGPlan$_g$ and other planners in solving the IPC4 and IPC3 benchmark suites. Each suite contains seven domains, with several variants in each. These variants address different features of PDDL2.2 that include versions on Strips, Strips with DP (derived predicates), temporal, temporal with TIL (deadlines), numeric, and complex (temporal and numeric). A complete description of each variant and its problem files can be found in the web sites of IPC4 and IPC3.

The International Planning Competition (IPC) is a biannual event started from 1998 and organized by the International Conference on Automated Planning and Scheduling. Each competition provides a set of planning problems from various domains for the participating planners to solve. In IPC4, all runs were carried out on the official computer of the competition, a 3-GHz two-CPU Linux PC. Following the rules of IPC4, all random planners set a fixed random seed, once and for all, throughout their experiments. Moreover, all planners must be fully automated, run with a fixed parameter setting for each instance attempted, and execute under a CPU time limit of 30 minutes and a main memory limit of 1 Gbytes.

Table 4.2 summarizes the number of instances solved by some of the best planners in

Table 4.2: Number of instances in each domain solved by the top planners that participated in IPC4.

Domain	# Instances	SGPlan$_g$	LPG-TD	Downward	Diag-Downward	Macro-FF	YAHSP	Crikey
Airport	200	155	134	50	50	21	36	64
Pipesworld	260	166	113	60	80	62	93	111
Promela	272	167	83	83	83	38	42	13
PSR	200	122	99	131	131	32	48	29
Satellite	288	207	157	36	36	36	–	–
Settlers	20	19	13	–	–	–	–	–
UMTS	300	274	200	–	–	–	–	–
Total	1540	1110	799	360	380	189	219	217

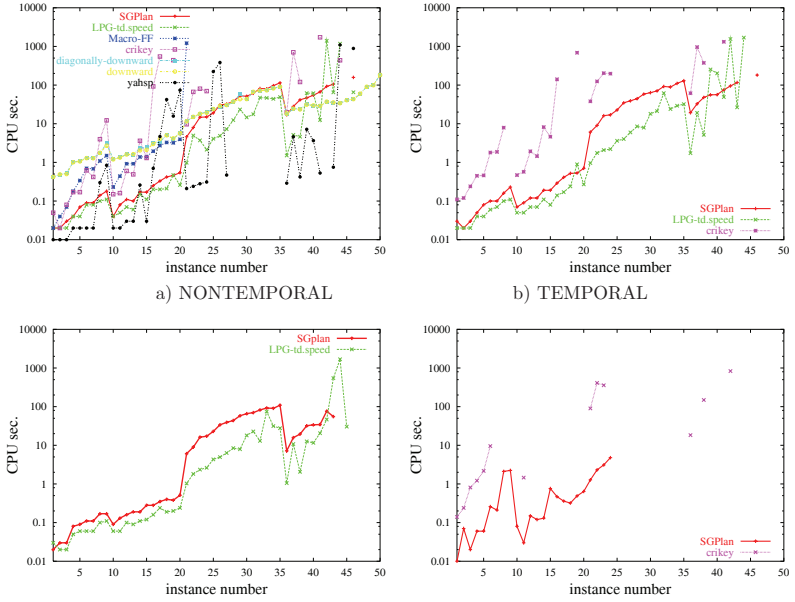


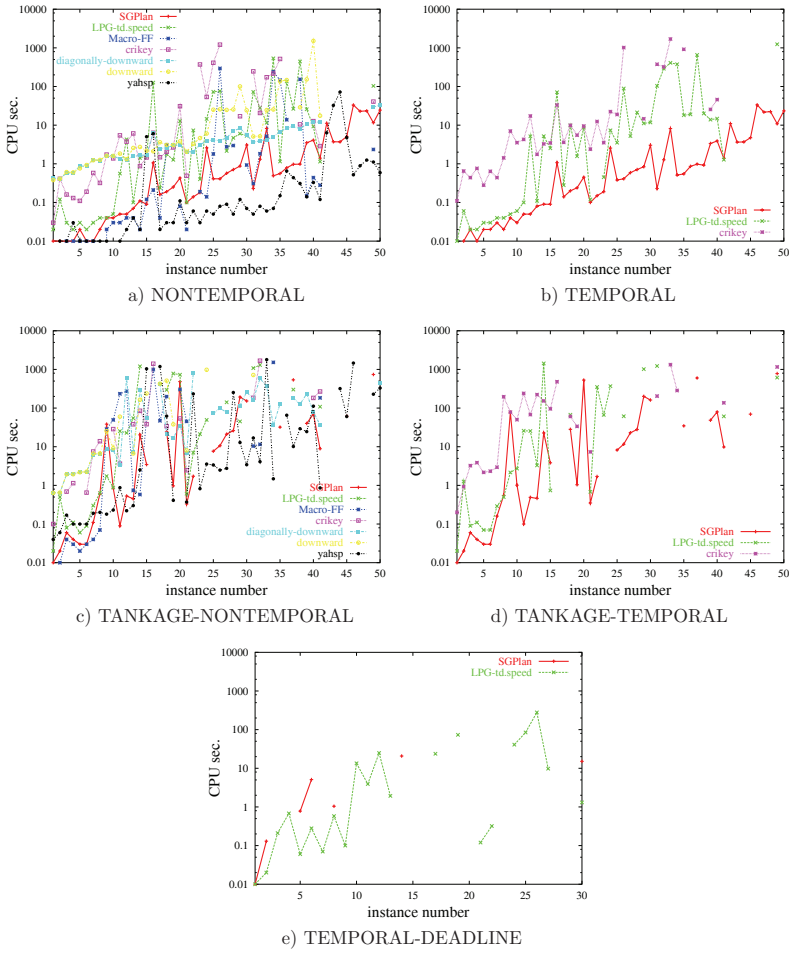
Figure 4.10: Comparison of the performance of IPC4 planners on the Airport domain.

each IPC4 domain. Figures 4.10-4.17 further depict the complete results on these planners.

Figure 4.10 shows the performance in solving the instances of the Airport domain. SGPlan_g can solve more instances than other planners in the TEMPORAL and the TEMPORAL-TIMEWINDOWS-COMPILED variants. In the NONTEMPORAL variant, SGPlan_g can solve 44 instances but not those numbered 44, and 46 to 50; whereas Downward, the leading planner for this variant, can solve all 50 instances. SGPlan_g cannot solve the five largest instances because some of the partitioned subproblems are too large and cannot be evaluated by the modified Metric-FF planner embedded in SGPlan_g. An obvious solution is to employ a more efficient basic planner when it is available. In fact, this is one of the major benefits of our partition-and-resolve approach. Another solution is to develop new partitioning methods so that the complexity of each subproblem is low enough to be handled by our modified Metric-FF planner. The design of these partitioning methods is open at this time.

Figure 4.11 shows that SGPlan_g can solve more instances in the NONTEMPORAL, TEMPORAL, TANKAGE-NONTEMPORAL, and TANKAGE-TEMPORAL variants of the Pipesworld domain than other planners. Further, SGPlan_g consistently has the shortest solution time in the TEMPORAL, TANKAGE-NONTEMPORAL, and TANKAGE-TEMPORAL variants. In the NONTEMPORAL variant, YAHSP and SGPlan_g are the only two planers that can solve all 50 instances. YAHSP, however, has the shortest solution time in most cases, although the difference is generally within one order of magnitude. Last, as is discussed in Section 4.1.3, SGPlan_g is not competitive in the TEMPORAL-DEADLINE variant. It can only solve two instances using strategy S_A and eight instances using S_B.

Figure 4.12 shows, for the Promela domain, that SGPlan_g can solve the most number of instances in the OPTICAL-TELEGRAPH-FL, PHILOSOPHERS, PHILOSOPHERS-DERIVEDPRED, and PHILOSOPHERS-FLUENTS variants. Further, SGPlan_g is the fastest planner in three of the variants and is slightly slower than YAHSP in PHILOSOPHERS.

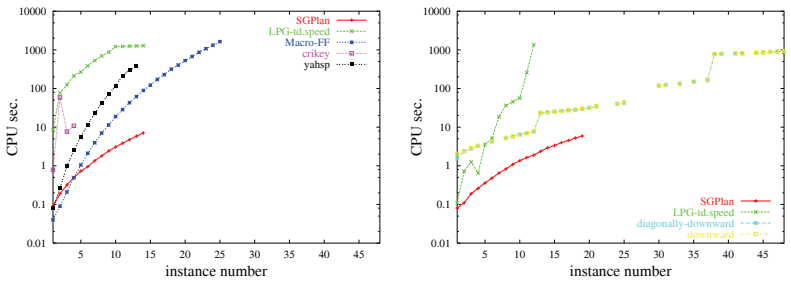


a) NONTEMPORAL b) TEMPORAL

c) TANKAGE-NONTEMPORAL d) TANKAGE-TEMPORAL

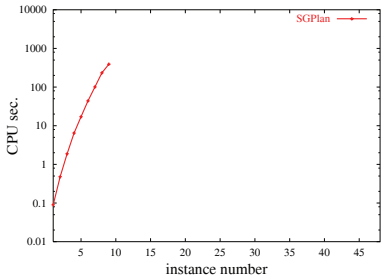
e) TEMPORAL-DEADLINE

Figure 4.11: Comparison of the performance of IPC4 planners on the Pipesworld domain.

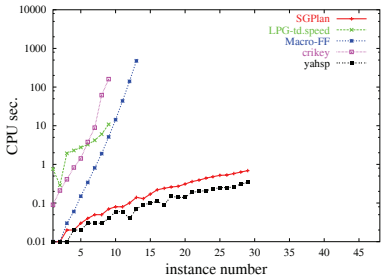


a) OPTICAL-TELEGRAPH

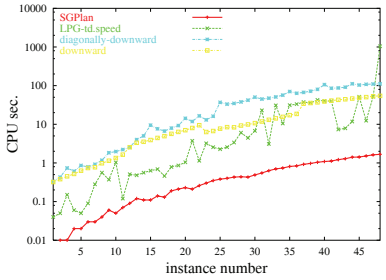
b) OPTICAL-TELEGRAPH-DP



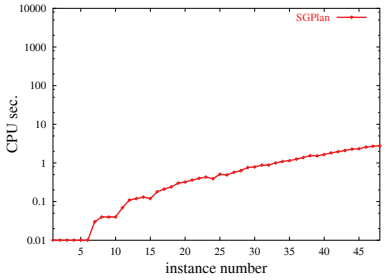
c) OPTICAL-TELEGRAPH-FL



d) PHILOSOPHERS



e) PHILOSOPHERS-DERIVEDPRED



f) PHILOSOPHERS-FLUENTS

Figure 4.12: Comparison of the performance of IPC4 planners on the Promela domain.

In the OPTICAL-TELEGRAPH and OPTICAL-TELEGRAPH-DP variants, the organizer of IPC4 provided two versions, one written in pure Strips and another in ADL. However, there are only 14 (resp., 19) instances in Strips and 48 (resp., 48) instances in ADL for the OPTICAL-TELEGRAPH (resp., OPTICAL-TELEGRAPH-DP) variant. There are more instances available in ADL because ADL is space-efficient in its problem representation, whereas instances specified in Strips require large files. (For example, the file size of OPTICAL-TELEGRAPH-14 is 38 Kbytes in ADL and 8.3 Mbytes in Strips.) Since SGPlan$_g$ cannot handle ADL at this time, it only solved the instances in pure Strips in these two variants. It was able to solve all the instances available in Strips and was the fastest in all these instances. However, other planners, such as Macro-FF and Downward, can handle instances in ADL and were able to solve more instances in these two variants.

Figure 4.13 shows that SGPlan$_g$ is the only planner that can solve some instances of all four variants of the PSR domain. In the SMALL variant, SGPlan$_g$, LPG, and Crikey have comparable performance and cannot solve the few largest instances. Like the AIRPORT-TEMPORAL variant, SGPlan$_g$ has difficulty with the few largest instances because its basic planner cannot handle the partitioned subproblems. In the MIDDLE variant, SGPlan$_g$, LPG, and Downward can solve all 50 instances. The situation in the MIDDLE-COMPILED and LARGE variants are similar to that in the OPTICAL-TELEGRAPH and the OPTICAL-TELEGRAPH-DP variants of the Promela domain. In these variants, Macro-FF and Downward can handle directly the ADL format, but SGPlan$_g$ must expand the ADL syntax to pure Strips and exhausted its memory limit when evaluating larger instances. We plan to extend SGPlan$_g$ to handle ADL directly in the future.

Figure 4.14 and Figure 4.15 show that SGPlan$_g$ can solve the most number of instances in seven variants of the Satellite domain. In the eighth variant (TIME), SGPlan$_g$ was not able to solve the few largest instances because its memory usage exceeded 1 Gbytes. In all

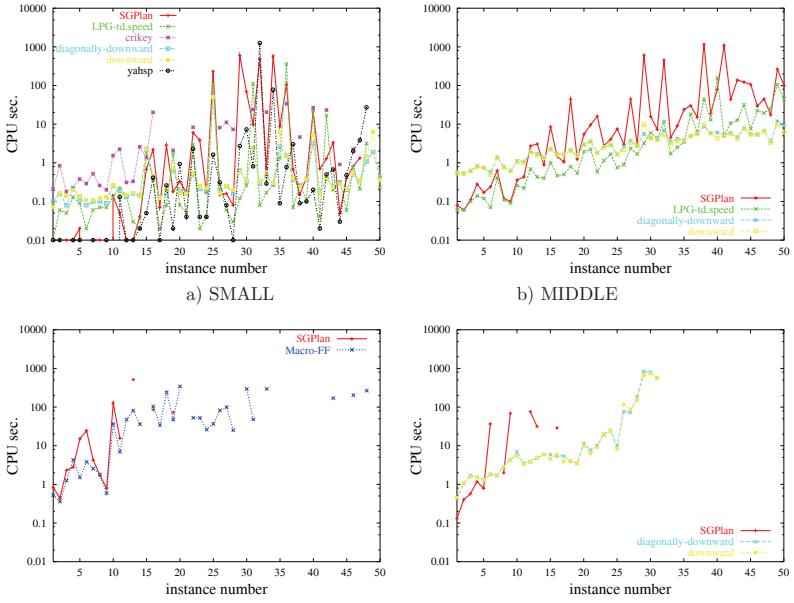


Figure 4.13: Comparison of the performance of IPC4 planners on the PSR domain.

the variants, SGPlan_g is the fastest planner except in the STRIPS variant.

Figure 4.16 shows that SGPlan_g can solve the most number of instances in all the six variants of the UMTS domain and is the fastest in four of them. SGPlan_g, however, is slower than LPG-TD in the FLAW and FLAW-TIL variants. The performance degradation of SGPlan_g in these variants is attributed to its implementation overhead, since many of their instances are easy and can be solved within five seconds of CPU time.

Figure 4.17 shows that SGPlan_g can solve all the instances in the Settlers domain except the 8^{th} instance, which we learned from the IPC4 organizers that it is an infeasible instance. SGPlan_g is also the fastest among all the planners.

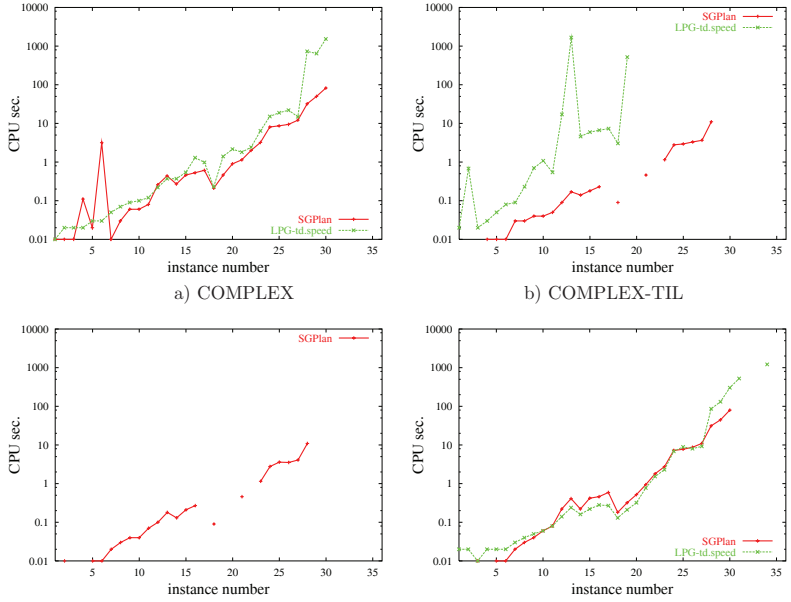
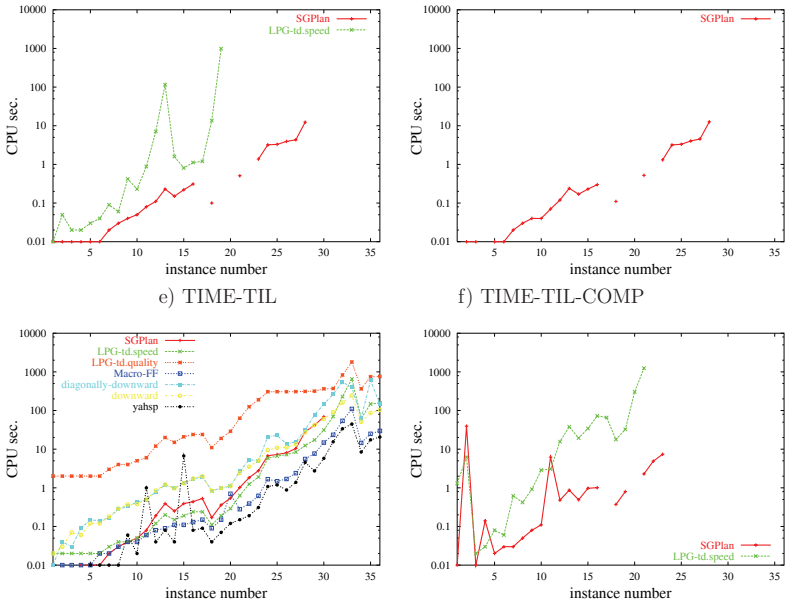


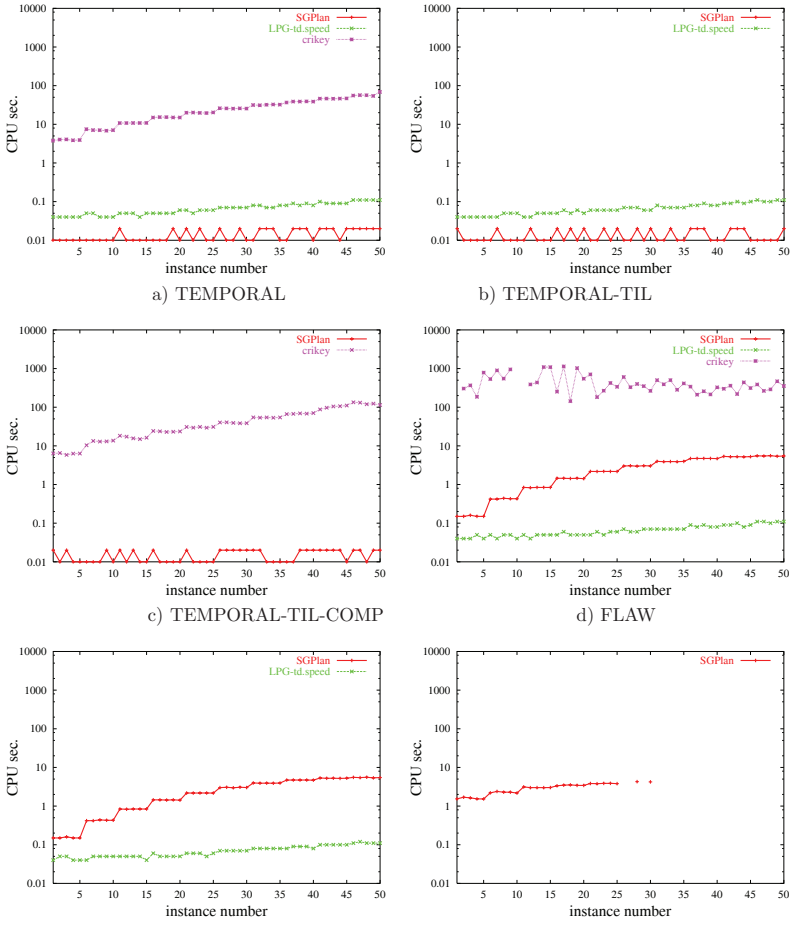
Figure 4.14: Comparison of the performance of IPC4 planners on the Satellite domain.



e) TIME-TIL f) TIME-TIL-COMP

g) STRIPS h) NUMERIC

Figure 4.15: Comparison of the performance of IPC4 planners on the Satellite domain.



a) TEMPORAL b) TEMPORAL-TIL

c) TEMPORAL-TIL-COMP d) FLAW

e) FLAW-TIL f) FLAW-TIL-COMP

Figure 4.16: Comparison of the performance of IPC4 planners on the UMTS domain.

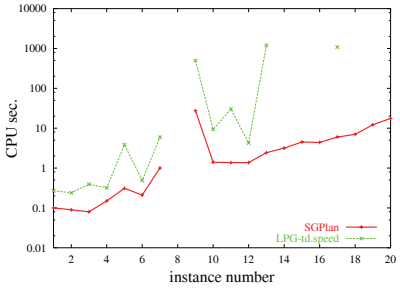


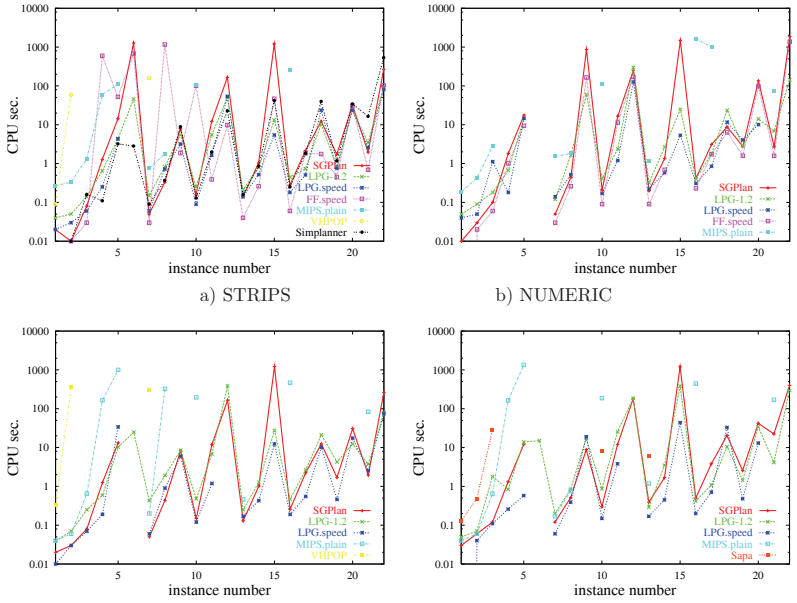
Figure 4.17: Comparison of the performance of IPC4 planners on the Settlers domain.

Table 4.3: Number of instances in each IPC3 domain solved by the eight planners compared.

Domain	# Instances	$SGPlan_g$	LPG-1.2	LPG.speed	FF.speed	MIPS.plain	VHPOP	Sapa	Simplanner
Depots	88	85	87	77	42	42	6	5	22
DriverLog	100	90	100	98	46	77	28	14	11
ZenoTravel	80	80	80	76	40	77	26	15	20
Rovers	80	52	35	33	29	29	27	11	9
Satellite	120	118	114	69	54	90	34	35	17
Settlers	20	19	0	0	0	0	0	0	0
FreeCell	20	20	2	18	20	0	1	0	12
Total	508	464	418	371	237	315	122	80	91

In addition to the IPC4 domains, we have evaluated the performance of $SGPlan_g$ on all the IPC3 domains. We have also downloaded the most recent version of LPG, LPG1.2. Our tests of the IPC3 instances using $SGPlan_g$ and LPG1.2 were conducted on our local computer, an AMD Athlon MP2000 PC with Linux Redhat 7.2 and 1-Gbyte main memory. For the other planners, we have used the competition results from the IPC3 Web site (`http://planning.cis.strath.ac.uk/competition/`). These results are slightly off from the results collected on our local computer because they were collected on an AMD Athlon MP1800 computer with 1-Gbyte main memory.

Table 4.3 summarizes the number of instances in each domain solved by the seven top planners. Overall, $SGPlan_g$ was able to solve the most number of instances than the other



a) STRIPS b) NUMERIC

c) SIMPLETIME d) TIME

Figure 4.18: Comparison of the performance of various planners on the Depots domain.

planners.

Figures 4.18-4.22 present the performance of the eight planners on the seven IPC3 domains. The following are some of the observations on these graphs. a) In the Depots and DriverLog domains, $SGPlan_g$ is generally the third fastest planner, next to FF.speed and LPG.speed. b) In the ZenoTravel domain, $SGPlan_g$ is the fastest in the SIMPLETIME and TIME variants. It is, however, the second fastest in the STRIPS and NUMERIC variants, where FF.speed is slightly faster. c) In the Freecell domain, $SGPlan_g$ and FF.speed are the only two planners that can solve all the instances. FF.speed, however, is faster. d) In the Settlers domain, $SGPlan_g$ is the single best planner. e) In the Rovers domain, $SGPlan_g$

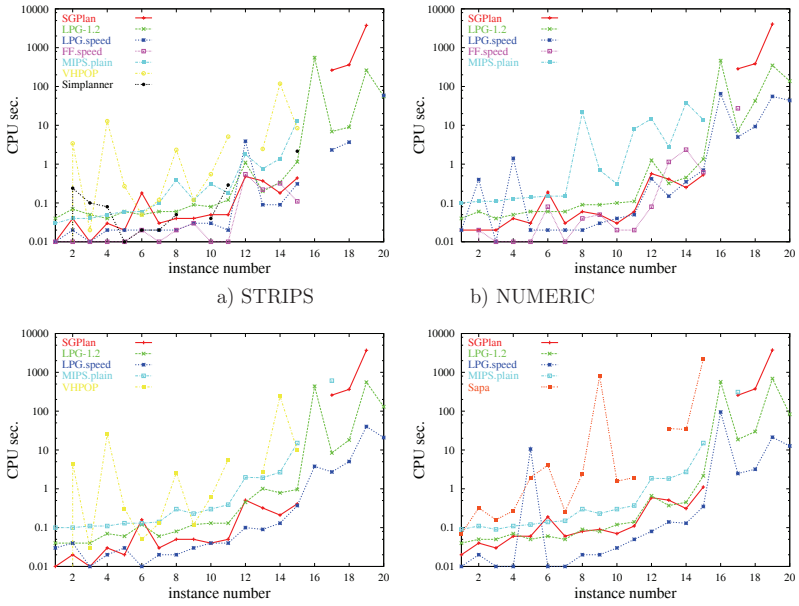


Figure 4.19: Comparison of the performance of various planners on the IPC3 DriverLog domain.

solvers significantly more instances than all other planners. It is only slightly slower than FF.speed in some instances and is faster than others. f) In the Satellite domain, SGPlan$_g$ solves 118/120 instances, more than any other planners. Only LPG1.2 that solves 114/120 is close to SGPlan$_g$. In terms of speed, SGPlan$_g$ is faster than LPG1.2 in the HARDNUMERIC, NUMERIC, and COMPLEX variants, but is slightly slower in the other three variants.

In summary, we have presented in this section SGPlan$_g$, a planner that won the first prize in the Suboptimal Temporal Metric Track and the second prize in the Suboptimal Propositional Track in IPC4. Our approach is based on the observation that the fraction of active global mutual-exclusion constraints across subproblems is very small, when the

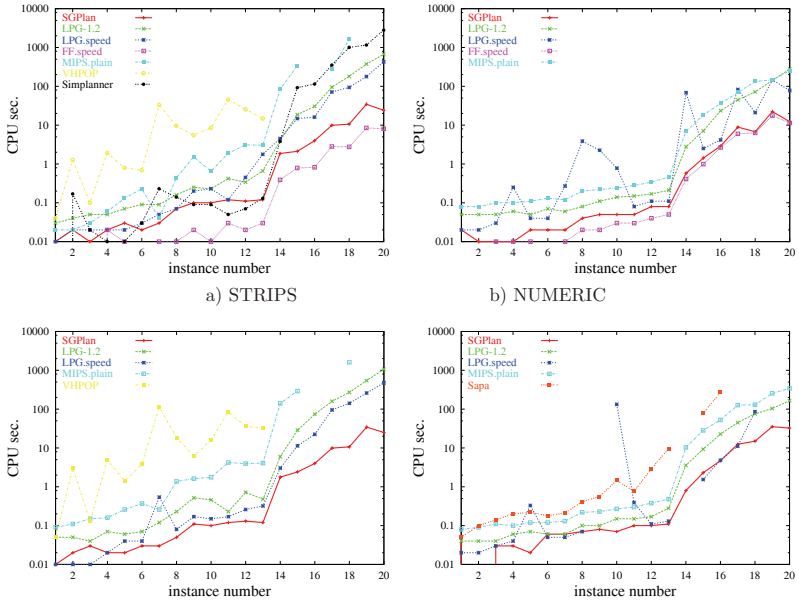
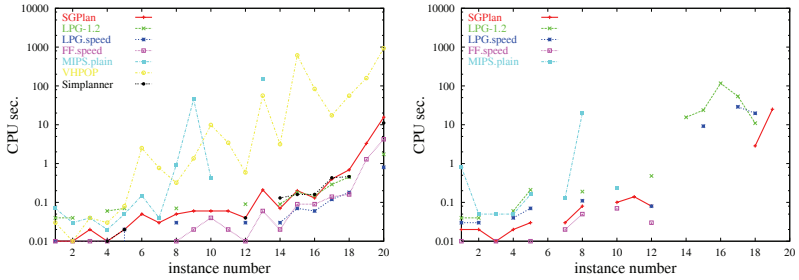
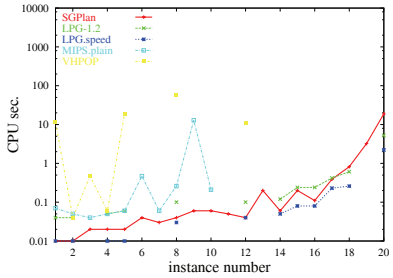


Figure 4.20: Comparison of the performance of various planners on the IPC3 ZenoTravel domain.



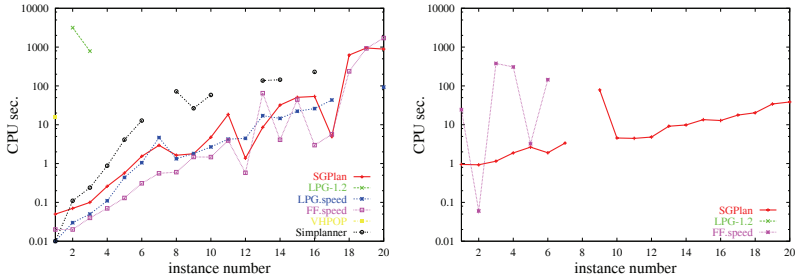
a) STRIPS b) NUMERIC



c) SIMPLETIME

Figure 4.21: Comparison of the performance of various planners on the IPC3 Rovers domain.

constraints of a planning problem are partitioned by its subgoals into subproblems. Based on the theory of extended saddle points, the partition-and-resolve approach can limit the search space to be backtracked in resolving violated global constraints. We have also discussed other related techniques in SGPlan$_g$ for reducing the search space and for handling the new features in PDDL2.2. Our experimental results show that SGPlan$_g$ performs well on both the IPC3 and IPC4 domains.



a) Freecell domain b) Settlers domain

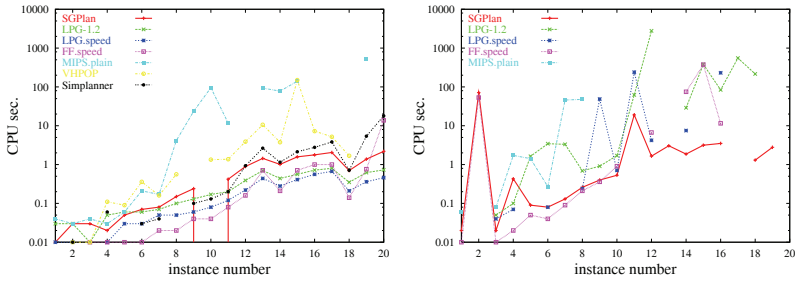
Figure 4.22: Comparison of the performance of various planners on the IPC3 Freecell and Settlers domains.

4.2 Applications on ASPEN Planning

In this section, we describe the ASPEN (Automated Scheduling and Planning Environment) system [16] developed at the Jet Propulsion Laboratory and its available benchmarks on spacecraft operation planning. We then present our prototype planner SGPlan$_t$(ASPEN,N) that partitions a problem along its temporal horizon into N stages, that calls ASPEN to solve the subproblems, and that resolves violated global constraints. Finally, we compare the performance between ASPEN and SGPlan$_t$(ASPEN,N).

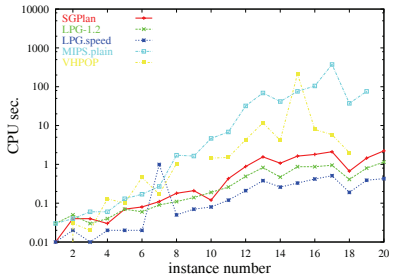
4.2.1 SGPlan$_t$(ASPEN): A planner using ASPEN to solve partitioned problems

ASPEN [16] is an objective-based planning system for the automated planning and scheduling of complex spacecraft operations. The application [45] involves finding a sequence of low-level commands from a set of high-level science and engineering goals, such as spacecraft operability constraints, flight rules, hardware models, science experiment goals, and operation procedures, subject to parameter dependencies and temporal and resource constraints.

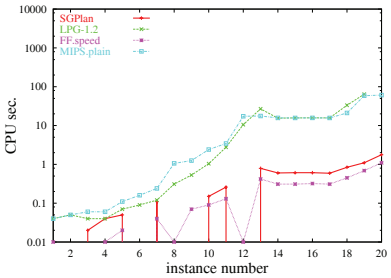


a) STRIPS

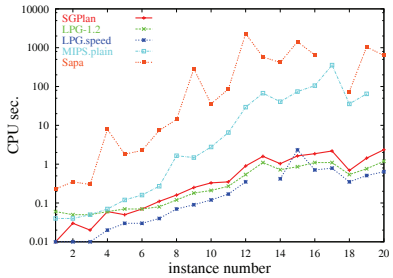
b) NUMERIC



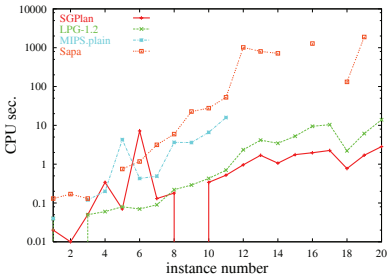
c) SIMPLETIME



d) HARDNUMERIC



e) TIME



f) COMPLEX

Figure 4.23: Comparison of the performance of various planners on the IPC3 Satellite domain.

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 in 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 optimize plan quality and search for feasible plans at the same time, it alternates between a repair phase and an optimization phase. In the repair phase [69], 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 resolve 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 a 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).

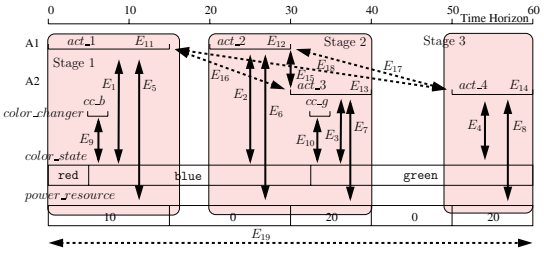
The ASPEN software can be tested on several publicly available benchmarks that schedule parallel spacecraft operations. In this thesis, we have tested the four available benchmarks in the public domain: a) The CX1-PREF benchmark [95] models the planning of operations of the Citizen Explorer-1 (CX-1) 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 [68] 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.

Figure 4.24 illustrates a toy problem in ASEPN [16] model and its constrained formulation. The problem involves scheduling four activities: *act_1* 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. 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 $\{E_{16}, E_{17}, E_{18}, E_{19}\}$.

In a MINLP formulation of the toy example, each of the nineteen constraints E_1-E_{19} in Figure 4.24 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 denoting 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`); $p(t)$ 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:


```
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}).

Figure 4.24: A toy example from ASPEN [16] 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 used. The number of iterations to solve the problem is reduced from 16 taken by ASPEN to 12 after partitioning.

(c1) $w(t) \leq p(t) \leq 25, \ \forall\, t = 0, \ldots, 60;$
// *power_resource* capacity constraint

(c2) $0 \leq s(act\_3) - e(act\_1) \leq 30;$
// *act_1* ends_before start of *act_3* by [0,30]

(c3) $s(act\_1) = t \implies c(t) - 2 = 0; \ \forall\, t = 0, \ldots, 60;$
// *color_state* constraint for *act_1*

(c4) $s(cc\_b) = t \implies c(t) - 1 = 0; \ \forall\, t = 0, \ldots, 60;$
// *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 $A \Longrightarrow B$, where A and B are equality or inequality constraints, can be encoded as an equivalent equality constraint $H(A \Longrightarrow B) = 0$:

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

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

4.2.2 Temporal locality in ASPEN planning

In this research, we analyze the constraints of a temporal planning problem 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 stage and that to resolve violated global constraints is complex and unknown. A good partitioning algorithm must achieve a proper balance between the overheads of local evaluation and global resolution.

In this research, we exploit the temporal locality of constraints in ASPEN 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.

To illustrate the temporal locality of constraints in ASPEN planning problems, Figure 4.24 (*resp.* Figure 4.25) shows the nineteen (*resp.* 3,687) constraints of an initial infeasible schedule generated by ASPEN [16] in solving the toy example (*resp.* CX1-PREF with sixteen orbits). After partitioning the constraints into three (*resp.* four) stages, the

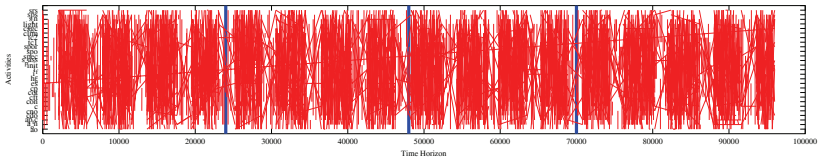


Figure 4.25: The 3,687 constraints of an initial infeasible schedule generated by ASPEN in solving CX1-PREF with 16 orbits. Each constraint is shown as a line that relates two activities (labeled in the y-axis) scheduled at two time instances in the horizon (x-axis). The partitioning of the constraints into four stages (separated by bold vertical lines) leads to 3,580 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 1,102 after partitioning.

resulting problem has fifteen (*resp.* 3,580) local constraints and four (*resp.* 107) global constraints. Since some global constraints may be 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.

If the number of stages is small, the overhead of resolving a small number of global constraints will be low, but the overhead of evaluating each stage will be high. On the other hand, if the number of stages is large, then the resolution of the many global constraints will be costly, even though the overhead of evaluating a stage will he low.

Next, we describe a partition-and-resolve procedure that implements constraint partitioning in the ASPEN planning system. The procedure iterates between calling a modified ASPEN planner to solve the constraint-partitioned subproblems, and using a constraint-reweighting strategy to resolve the violated global constraints across the subproblems. We demonstrate significant improvements in solving some ASPEN benchmarks for aerospace operations. For example, the problem in Figure 4.24 (*resp.* 4.25) can be solved by ASPEN in 16 (*resp.* 12,043) iterations and by our our implementation in 12 (*resp.* 1,102) iterations with the same (*resp.* better) quality.

```
1.  procedure SGPlan_t(ASPEN,N)
2.      generate initial plan and set initial temperature T;
3.      partition time horizon into N stages;
4.      repeat
5.          num_descents ← 1;
6.          for t = 1 to N
7.              for k = 1 to num_descents
8.                  call ASPEN to solve (3.31) by generating a new schedule in a child process;
9.                  evaluate Γ_d(t) and the Metropolis probability controlled by T;
10.                 if Γ_d(t) is accepted then
11.                     call ASPEN to apply the action in the main process;
12.                     update penalties α(t) and β(t) on violated local constraints;
13.                 end_if
14.             end_for
15.         end_for
16.         update penalties γ and η on violated global constraints;
17.         num_descents ← min(100, num_descents * 2);
18.         reduce temperature T ← T × c;
19.         dynamically repartition the stages (only done in SGPlan_t(ASPEN,N,DYN_P));
20.     until no change in z, α, β, γ, η in an iteration;
21. end_procedure
```

Figure 4.26: SGPlan$_t$(ASPEN,N): 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, and that resolves violated global constraints. Annealing (Lines 9-10) is used to probabilistically accept a probe with worse penalty-function value during descents of Γ_d.

4.2.3 Implementation of partition-and-resolve search in ASPEN.

Based on the pseudo code in Figure 5.2, we have implemented SGPlan$_t$(ASPEN,N), a planner that partitions a planning problem along its temporal horizon into N subproblems of the form in (3.31) and that calls ASPEN to solve the subproblems [15]. In our implementation, we set the weight of the objective function $J(z)$ in (3.31) to 100 (since the preference score is between 0 to 1), initialize all penalties to zeroes, and increase the penalties of violated schedule constraints in each iteration by 0.1.

In generating a new schedule from the current schedule during descents of Γ_d (Line 8 of Figure 4.26), 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 infeasible local minima. To this end, SGPlan$_t$(ASPEN,N) 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_d based on the Metropolis probability, with the acceptance probability decreasing as the temperature decreases. In our algorithm, we fix the initial temperature to 1,000 and reduce it in every iteration by a factor $c = 0.8$.

Two other important issues that must be addressed in our partition-and-resolve implementation are the number of stages used and the duration of each. In ASPEN, a conflict has an active window bounded by a start time and an end time called the *time points*. Adjacent time points can be collapsed into a stage, since ASPEN has discrete time horizons.

We have studied both the static and the dynamic partitioning of stages. In static partitioning, SGPlan$_t$(ASPEN,N,STATIC$_P$) partitions the horizon statically and evenly into N stages. This simple strategy often leads to an unbalanced number of time points in different stages. During a search, some stages may contain no conflicts to be resolved, while others may contain a lot of conflicts. Such an imbalance leads to search spaces of different sizes across different stages and search times that may be dominated by those in a few stages.

To achieve a better balance of activities across stages, SGPlan$_t$(ASPEN,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, Figure 4.27 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

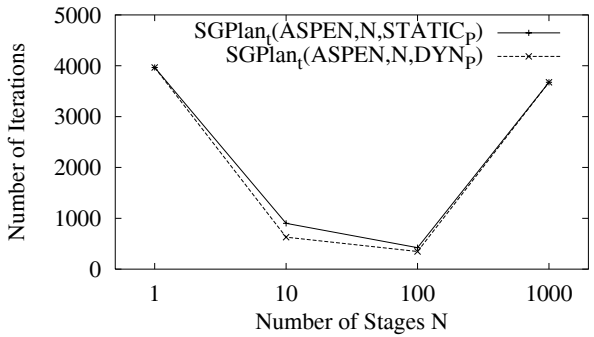
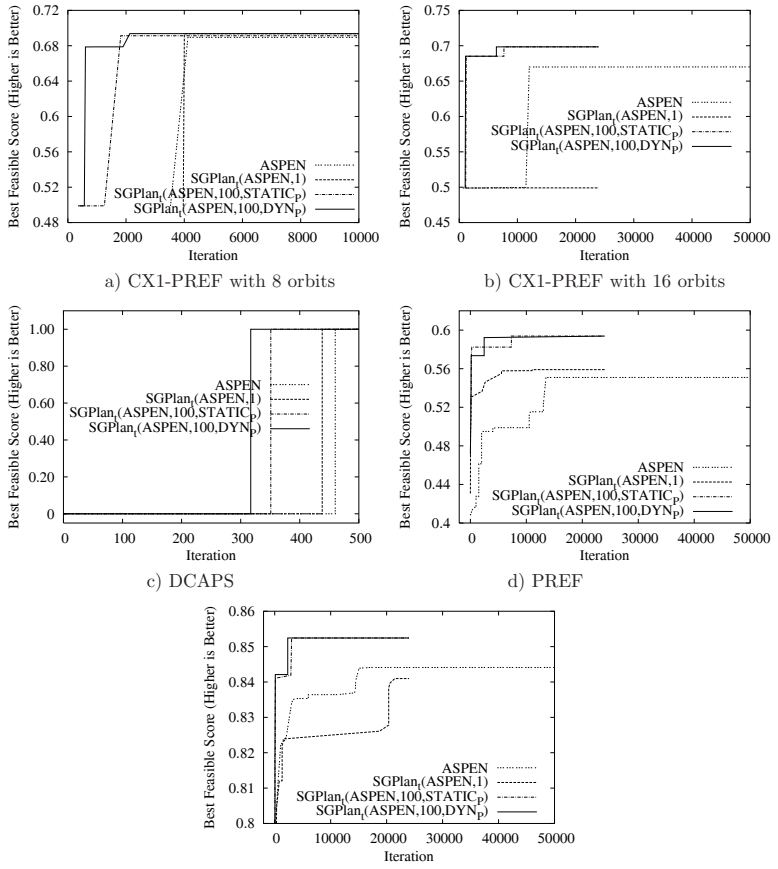


Figure 4.27: Number of iterations taken by static and dynamic partitioning to find a feasible plan in the 8-orbit CX1-PREF problem.

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.

4.2.4 Experimental results

Figure 4.28 compares the performance of ASPEN, SGPlan$_t$(ASPEN,100), and SGPlan$_t$(ASPEN,1) (a version of our planner without partitioning) on the five 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$_t$(ASPEN,100) is not guaranteed to find optimal schedules, it can find 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 SGPlan$_t$(ASPEN,100) is able to find schedules of the same quality one to two orders faster than ASPEN and SGPlan$_t$(ASPEN,1) and much better schedules when they converge.



a) CX1-PREF with 8 orbits

b) CX1-PREF with 16 orbits

c) DCAPS

d) PREF

e) OPTIMIZE

Figure 4.28: Quality-time comparisons of ASPEN, SGPlan$_t$(ASPEN,1), SGPlan$_t$(ASPEN,100,STATIC$_P$), and SGPlan$_t$(ASPEN,100,DYN$_P$). (All runs involving SGPlan$_t$ were terminated at 24,000 iterations.)

4.3 Summary

We have presented in this chapter the applications of the constraint-partitioning approach to solve planning problems in PDDL2.2 domains and ASPEN domains.

For PDDL2.2 domains, we have observed that the fraction of active global mutual-exclusion constraints across subproblems is very small when the constraints of a planning problem are partitioned by its subgoals into subproblems. We have then presented the SGPlan_{g} planner that partitions the constraints of each PDDL2.2 planning problem by its subgoals and uses a heuristic planner Metric-FF as the basic solver for each subproblem. We have also discussed other related techniques in SGPlan_{g} for reducing the search space and for handling the new features in PDDL2.2. We have shown experimental improvement of SGPlan_{g} over existing planners on both the IPC3 and IPC4 domains, and have also analyzed the time-quality trade-off of SGPlan_{g}.

For ASPEN domains, we have observed temporal locality of the constraints and have proposed to partition the constraints by the time horizon. We have then presented the integration of the constraint partitioning approach with the original ASPEN system. We have described a global search strategy based on simulated annealing in order to resolve global inconsistencies and a dynamic partitioning strategy in order to balance the search overhead across different subproblems. Finally, we have shown significant performance improvement in terms of planning time and solution quality in solving some ASPEN benchmarks.

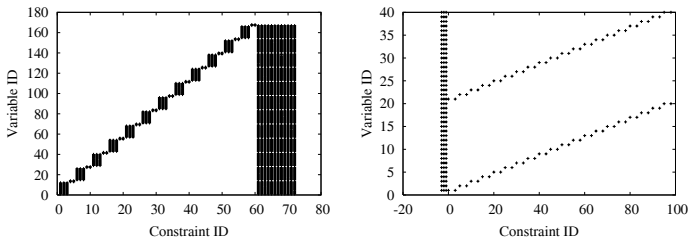
Chapter 5

Application on Mathematical Programming Benchmarks

In this chapter, we apply the constraint partitioning approach to solve some large MINLP and CNLP benchmarks. Based on our observation that MINLPs and CNLPs in many engineering applications have highly structured constraints, we partition these problems by their constraints into subproblems, solve each subproblem by an existing MINLP or CNLP solver, and resolve violated global constraints across subproblems using ESPC. Constraint partitioning allows many benchmark problems that cannot be solved by existing solvers to be solvable because it leads to easier subproblems that are significant relaxations of the original problem. We study various automated partitioning methods and strategies for resolving global constraints. We demonstrate the performance of our approach in solving some large-scale MINLP and CNLP benchmarks and show significant improvements in time and quality over those of existing solvers.

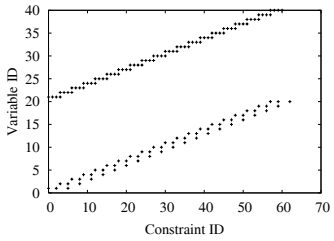
5.1 Problem Structure of Benchmarks

We have selected our MINLP benchmarks from the MacMINLP library [55], and CNLP benchmarks from the CUTE library [13]. There are 43 MINLP problems in MacMINLP from applications including nuclear core reloading optimization, optimal design of multiproduct batch plant, bar space truss design, optimal marketing of a new product in a multiattribute space, determination of optimum number of trays in a distillation column, minimizing total average cycle stock, trim loss minimization in paper industry, and engineering problems in

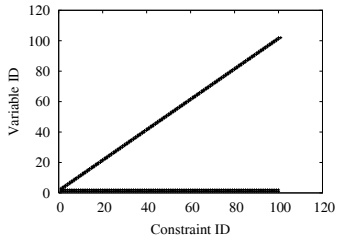


a) TRIMLON12 (MINLP)

b) POLGAS (MINLP)



c) OPTCDEG3 (CNLP)



d) ORTHRGDS (CNLP)

Figure 5.1: Regular structures of constraints in some MINLP and CNLP benchmarks. A dot in each graph represents a variable associated with a constraint.

machine design. The CUTE library includes more than 200 problems from many applications including structural design, optimal control, and engineering design.

We have observed that the constraints of many application benchmarks do not involve variables that are picked randomly from their variable sets. Invariably, many constraints in existing benchmarks are highly structured because they model spatial and temporal relationships that have strong locality, such as those in physical structures, optimal control, and staged processing.

Figure 5.1 illustrates this point by depicting the regular constraint structure of four

benchmarks. It shows a dot where a constraint (with unique ID on the x axis) is related to a variable (with a unique ID on the y axis). With the order of the variables and the constraints arranged properly, the figure shows a strong regular structure of the constraints.

Based on the regular constraint structure of a problem instance, we can cluster its constraints into multiple loosely coupled groups using constraint partitioning. An example to partition the TRIMLON12 problem into 12 subproblems by its index J is shown in Figure 1.2 in Section 1.2.1. Given a MINLP problem P_m defined in equation (1.1), the problem formulation under constraint partitioning is given in equation (3.23), which we rewrite below:

$$
\begin{aligned}
(P_t): \quad & \min_{z} \quad J(z) && && (5.1)\\
& \text{subject to} \quad h^{(t)}(z(t)) = 0, \quad g^{(t)}(z(t)) \le 0 && \text{(local constraints)}\\
& \text{and} \quad H(z) = 0, \quad G(z) \le 0 && \text{(global constraints).}
\end{aligned}
$$

where $h^{(t)} = (h_1^{(t)}, \ldots, h_{m_t}^{(t)})^T$ and $g^{(t)} = (g_1^{(t)}, \ldots, g_{r_t}^{(t)})^T$ are local constraints; and $H = (H_1, \ldots, H_p)^T$ and $G = (G_1, \ldots, G_q)^T$ are global constraints.

For example, the TRIMLON12 problem is partitioned into the following subproblems:

$$
\begin{aligned}
&\text{variables:} && y[j], m[j], n[j,i], \quad \text{where} \quad i = 1, \cdots, I, \ j \in S_k && \\
&\text{objective:} && \min_{z=(y,m,n)} f(z) = \textstyle\sum_{j=1}^{J} c[j] * m[j] + C[j] * y[j] && (\text{OBJ}_j)\\
&\text{local const.:} && B_{min} \le \textstyle\sum_{i=1}^{I} (b[i] * [n[i,j]) \le B_{max} && (\text{C1}_j)\\
& && \textstyle\sum_{i=1}^{I} n[i,j] - N_{max} \le 0 && (\text{C2}_j)\\
& && y[j] - m[j] \le 0 && (\text{C3}_j)\\
& && m[j] - M * y[j] \le 0 && (\text{C4}_j)\\
&\text{global const.:} && Nord[i] - \textstyle\sum_{j=1}^{J} (m[j] * n[i,j]) \le 0. && (\text{C5}_j)
\end{aligned}
$$

Out of the 72 constraints, 60 are local constraints and 12 are global constraints.

The keys to the success of using constraint partitioning to solve MINLPs and CNLPs, therefore, depend on the identification of the constraint structure of a problem instance and the efficient resolution of its violated global constraints. To this end, we study the following issues.

a) *Automated analysis of the constraint structure of a problem instance and its partitioning into subproblems.* We study to analyze the strcuture of an instance specified in some standard form (such as AMPL [24] and GAMS). We show methods for determining the structure of an instance after possibly reorganizing its variables and constraints, and identifying the dimension by which the constraints can be partitioned.

b) *Optimality of the partitioning.* The optimality relies on trade-offs between the number of global constraints to be resolved and the overhead for evaluating each subproblem. We propose a metric for comparing various partitioning schemes and a simple and effective heuristic method for selecting the optimal partitioning according to the metric.

c) *Resolution of violated global constraints.* We apply the theory of extended saddle points for resolving violated global constraints by formulating the NLPs into a penalty formulation and solving a modified subproblem with biased objective function at each partition. We develop a partition-and-resolve solver based on the general ESPC search framework proposed in Chapter 3. We study strategies for updating penalties of violated constraints in the ESPC theory for solving MINLPs and CNLPs.

5.2 Partitioning and Resolution Strategies

In this section, we study the strategies for automated partitioning and efficient resolution of inconsistent global constraints.


```
1. procedure CPOPT
2.     call automated_partition(); // automatically partition the problem //
3.     γ ⟵ γ0; η ⟵ η0; // initialize penalty values for global constraints//
4.     repeat        // outer loop //
5.         for t = 1 to N // iterate over all N stages to solve (3.31) in each stage //
6.             apply an existing solver to solve (3.31)
7.             call update_penalty(); // update penalties of global constraints //
8.         end_for;
9.     until stopping condition is satisfied
10. end_procedure
```

Figure 5.2: CPOPT: Implementation of the partition-and-resolve framework to look for CLM_m of (5.1).

5.2.1 CPOPT: the partition-and-resolve procedure

Figure 5.2 presents CPOPT, a partition-and-resolve procedure for solving the constraint-partitioned problem P_t in (5.1). It first partitions the constraints into N subproblems (Line 2 of Figure 5.2b, discussed in Section 5.2.2). With fixed γ and η, it then solves (3.31) defined in stage t using an existing solver (Line 6). To satisfy the differentiability requirement of the objective function in (1.1), we transform (3.31) into the following equivalent problem with a differentiable objective:

$$\begin{aligned} \min_{z(t)} \quad & J(z) + \gamma^T a + \eta^T b \\ \text{subject to} \quad & h^{(t)}(z(t)) = 0 \quad \text{and} \quad g^{(t)}(z(t)) \le 0, \\ & -a \le H(z) \le a \quad \text{and} \quad G(z) \le b \end{aligned} \tag{5.2}$$

where a and b are non-negative auxiliary vectors. After solving each subproblem, we increase the penalties γ and η on the violated global constraints (Line 7, discussed in Section 5.2.3). The process is repeated until a CLM_m to (3.23) is found or when γ and η exceed their maximum bounds (Line 9).

We describe below the partitioning of the constraints and the update of the penalties.

5.2.2 Strategies for partitioning constraints into subproblems

Our goal in Line 2 of Figure 5.2b is to partition the constraints in such a way that minimizes the overall search time. Since the enumeration of all possible ways of partitioning is computationally prohibitive, we restrict our strategy to only partitioning by the index vectors of problems modelled by the AMPL language [24].

Definition 2. An *index vector* V in an AMPL model is a finite ordered array of discrete elements that are used to index variables and constraints.

For example, TRIMLON12 described in Section 1.2.1 has two index vectors: $I = J = \{1, \cdots, 12\}$. A variable or a constraint function can be indexed by one or more index vectors: $n[i, j], i \in I, j \in J$, is indexed by I and J; and (C5) is indexed by I alone.

Definition 3. A *partitioning index vector (PIV)* of an AMPL model is an index vector in the model for partitioning the constraints.

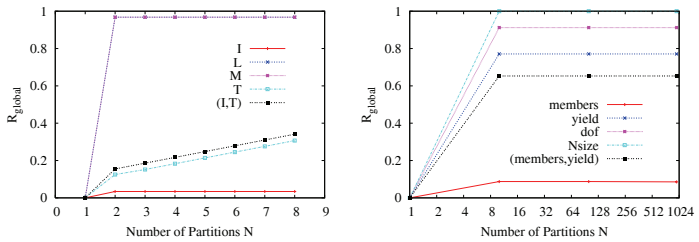
Definition 4. *Constraint partitioning by PIV.* Given a PIV of an AMPL model, an N-partition by PIV is a collection of subsets of the PIV, $S_1, \cdots, S_N$, where a) $S_i \in PIV$; b) $S_1 \cup \cdots \cup S_N = PIV$; and c) $S_i \cap S_j = \varnothing$ for $i \neq j, i, j = 1..N$.

The constraints of a problem can be partitioned along one or more index vectors. With multiple index vectors, the Cartesian-product space of the PIVs is partitioned into subsets. For instance, we have shown in Section 1.2.1 the partitioning of TRIMLON12 by J into $N = 12$ subproblems; that is, PIV $= \{J\}$, and $S_1 = \{1\}, \cdots, S_{12} = \{12\}$. This allows all the constraints indexed by J (C1 to C4) to be grouped into local constraints, and those not indexed by J (C5) to be the global constraints.

We argue that it is reasonable and effective to partition constraints by their index vectors. First, indexing is an essential mechanism in modeling languages like AMPL and GAMS for representing a complex problem in a compact form. Without it, it will be very cumbersome to use a unique name for each variable, especially when there are thousands of variables and constraints. Second, index vectors in large application problems are typically associated with physical entities. When constraints are organized and partitioned by index vectors, their partitioning can be interpreted meaningfully. For example, index vector J in TRIMLON12 corresponds to the possible cuts of paper rolls, and a subproblem partitioned by J entails the optimization of the paper production in each cut individually, while the overall production is globally constrained by the client orders.

Given a MINLP specified in AMPL, we present in the following our approach to automatically partition the problem by its constraints. We propose a metric to measure the quality of partitioning, present an algorithm to select the optimal PIV, illustrate trade-offs between the number of partitions and the overall complexity, and show an efficient heuristic for determining the optimal number of partitions.

a) **Metric of partition-ability.** We define R_{global} to measure the effectiveness of using PIVs for partitioning the constraints of a problem. Since the time to solve a partitioned problem is largely driven by the overhead in resolving its inconsistent global constraints, we define R_{global} to be the ratio of the number of global constraints to the total number of all constraints. This metric also needs to account for shared variables in multiple subproblems that must be consistent with each other. For simplicity, we assume each shared variable v that appears in k subproblems to be equivalent to $k - 1$ global constraints, where the i^{th} constraint involves the consistency between the i^{th} copy and the $i + 1^{st}$ copy. Note that the metric is heuristic because the exact overhead depends on the difficulty of resolving the inconsistent global constraints and not on the number of global constraints.



a) C-RELOAD-r-104 b) SPACE-960-r (MINLP)

Figure 5.3: Ratio of global constraints when partitioned by different PIVs for two MINLPs.

b) **Selection of PIV.** To select the best PIV that minimizes R_{global}, we observe from the benchmarks tested that the best PIV for a problem instance is independent of the number of partitions N. To illustrate this observation, Figure 5.3 plots the value of R_{global} for various PIVs as a function of N for two benchmarks. It shows that the best PIV that minimizes R_{global} is the same for all N. Based on this property, we first fix an arbitrary value of N in our implementation. As there are usually less than five index vectors in a model file, we just enumerate all possible combinations of PIVs, compute R_{global} for each case, and pick the one that minimizes R_{global}.

c) **Number of partitions.** Based on the best PIV selected, we decide next the number of partitions. Experimentally, we have observed a convex relationship between N and the total solution time. We illustrate this observation in Table 5.1 for various values of N on the SPACE-960-r MINLP from the MacMINLP library [55]. It shows the average time to solve a subproblem, the total time to solve all the subproblems in one iteration, the number of iterations needed to resolve the inconsistent global constraints, and the overall time to solve the problem. The best N for this problem is 30.

The convex relationship is intuitively reasonable. When the number of partitions is small

Table 5.1: Trade-offs between N and total solution time on the SPACE-960-r problem

Number of partitions N	1	15	30	60	120	240	480
Time per subproblem	>3600	8.4	3.3	3.1	2.8	2.7	2.6
Time per iteration	>3600	126	99	186	336	648	1248
Number of iterations	1	1	1	2	2	2	5
Total time to solve problem	>3600	126	99	372	672	1296	6240

1. **procedure optimal_number_of_partitions** (PIV)
2. $N \longleftarrow |PIV|$; $last\_time \longleftarrow \infty$;
3. **repeat**
4. Evaluate a subproblem under N partitions, record the solution time $T_p(N)$;
5. $overall\_time \longleftarrow T_p(N) \times N$;
6. **if** ($overall\_time > last\_time$) **then return** ($2N$);
7. $last\_time \longleftarrow overall\_time$;
8. $N \longleftarrow N/2$;
9. **end_repeat**
10. **end_procedure**

Figure 5.4: An iterative algorithm to estimate the optimal number of partitions.

or when there is no partitioning, each subproblem is large and expensive to evaluate, although the global constraints will be few in number and easy to revolve. In the other extreme, when there are many partitions, there will be many global constraints that are hard to resolve, although each subproblem is small and easy to evaluate.

The convex relationship allows us to determine an optimal number of partitions that minimizes the overall solution time. The idea is to start with the maximum number of partitions in the original problem (Line 2 of Figure 5.4) and evaluate a few subproblems there in order to estimate $T_p(N)$, the average time to solve a subproblem when there are N partitions (Line 4). We also evaluate *overall_time*, the time to solve all the subproblems once (Line 5). Assuming the number of iterations for resolving the global constraints to be small, *overall_time* will be related to the time to solve the original problem by a constant factor. This assumption is generally true for the benchmarks tested when N is close to the optimal value (as illustrated in Table 5.1). Next, we reduce N by half (Line 8) and repeat the process. We stop the process when we hit the bottom of the convex curve and have

Table 5.2: Average and standard deviation of solution time per subproblem for two benchmarks.

Problem instance	ORTHRGDS	ORTHRGDS	SPACE-960-r	SPACE-960-r
Number of partitions N	1000	20	100	10
Avg. time per subproblem ($T_p(N)$)	1.8	8.5	2.8	9.4
Std. dev. of time per subproblem	0.021	0.31	0.013	0.015

found the number of partitions that leads to the minimum *overall_time* (Line 6).

The algorithm requires $T_p(N)$, which can be estimated accurately based on the observation that it has little variations when the constraints are partitioned evenly. Table 5.2 illustrates this observation and shows that the standard deviation of the time to evaluate a subproblem is very small for two values of N. As a result, we only evaluate one subproblem in each iteration of Figure 5.4 in order to estimate $T_p(N)$ (Line 4).

For the SPACE-960-r example in Table 5.1, we set N as 480, 240, 120, 60, 30, 15 and stop at $N = 15$ because *overall_time* increases from $N = 30$ to $N = 15$. We finally choose $N = 30$. The total time to get this estimate after solving 6 subproblems is only 22.9 seconds, which is small when compared to the 160.45 seconds required for solving the original problem (Table 5.3).

5.2.3 Strategies for updating penalty values

After solving each subproblem, we use the following rule to update the penalty vectors γ and μ of violated global constraints (Line 7 of Figure 5.2b):

$$\gamma \longleftarrow \gamma + \rho^T |H(z)|, \qquad \eta \longleftarrow \eta + \varrho^T \max(0, G(z)), \tag{5.3}$$

where ρ and ϱ are vectors for controlling the update rate of γ and η.

We update each element of ρ and ϱ dynamically until the corresponding global constraint is satisfied. Vector ρ is initialized to ρ_0 and is updated as follows. For each global constraint

H_i, $i = 1, \cdots, p$, we use c_i to count the number of consecutive subproblem evaluations in which H_i is violated since the last update of ρ_i. After solving a subproblem, we increase c_i by 1 if H_i is violated; if c_i reaches threshold K, which means that H_i has not been satisfied in K consecutive subproblem evaluations, we increase ρ_i by:

$$\rho_i \longleftarrow \rho_i \times \alpha, \qquad \text{where } \alpha > 1, \tag{5.4}$$

and reset c_i to 0. If H_i is satisfied, we reset ρ_i to ρ_0 and c_i to 0. In our implementation, we choose $\rho_0 = 0.01$, $K = 3$ and, $\alpha = 1.25$. We update ϱ in the same manner.

The procedure in Figure 5.2 may generate fixed points of the ℓ_1-penalty function in (3.15) that do not satisfy (3.16). This happens because an ESP is a local minimum of (3.15) but not the converse. One way to escape from infeasible fixed points of (3.15) is to allow periodic decreases of γ and η (Line 7 of Figure 5.2b). The goal of these decreases is to "lower" the barrier in the penalty function in order for local descents in the inner loop to escape from an infeasible region. In our search, we scale down γ and η by multiplying each penalty value by a factor randomly generated between 0.4 and 0.6 if we cannot decrease the maximum violation of global constraints or improve the objective quality after solving five consecutive subproblems.

5.3 Experimental Results

In this section, we compare the performance of CPOPT to that of other leading solvers. In CPOPT, if a partitioned subproblem defined in (3.31) is a MINLP, CPOPT first generates a good starting point by solving it as a CNLP using SNOPT [32], while ignoring the integrality of integer variables. It then applies MINLP_BB [54] to solve the subproblem. If the partitioned subproblem is a CNLP, CPOPT applies SNOPT to solve it.

We have compared CPOPT to two of the best MINLP solvers, MINLP_BB [54] and

BARON [76], on a collection of MINLP benchmarks from the MacMINLP library [55]. MINLP_BB implements a branch-and-bound algorithm with a sequential-quadratic-programming (SQP) solver for solving continuous subproblems, whereas BARON is a mixed-integer constrained solver implementing the branch-and-reduce algorithm. The complete results on MacMINLP benchmarks are reported in Table 5.3. For each problem, we show the number of constraints n_c, the number of variables n_v, and the solution quality and the solution time (in seconds) for each solver. We can see that CPOPT is faster than BARON and MINLP_BB in most large problems where it takes more than 10 seconds to solve by CPOPT. It can also solve some large problems that cannot be solved by BARON and MINLP_BB, such as TRIMLOSS12.

We have also compared CPOPT to two of the best CNLP solvers, Lancelot [17], an augmented Lagrangian method, and SNOPT [32], an SQP solver, on the CNLPs from the CUTE library [13]. We have shown the results on large CUTE benchmarks that cannot be solved by either SNOPT or Lancelot in Table 5.5. The results show that CPOPT can find the best solution for most test problems, that it is one to two orders of magnitude faster, and that it is able to solve large problems that no other solver can tackle. The complete results on other CUTE benchmarks in Table 5.7. On these small problems that are easy to solve, the three solvers have same solution quality for most problems. CPOPT is usually slower than SNOPT due to partitioning overhead and slightly faster than LANCELOT for the small problems in Table 5.7.

Table 5.3: Results on solving MINLP benchmarks from the MacMINLP library [55]. Results on MINLP_BB and BARON were obtained by submitting jobs to the NEOS server [61] and BARON's site [76], respectively; results of other solvers were collected on an AMD Athlon MP2800 PC running RH Linux AS4 and a time limit of 3,600 sec. All timing results are in seconds and should be compared within a solver but not across solvers because they might be run on different computers. Solutions with the best quality are boxed. "–" means that no feasible solutions were found in the time limit.

Test Problem			MINLP_BB		BARON		CPOPT(MINLP_BB)	
ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
BATCH	73	46	2.85E5	0.58	2.59E5	2.03	2.59E5	1.92
C-RELOAD-14a	308	342	-1.01	1.43	-1.01	0.98	-1.01	2.35
C-RELOAD-14b	308	342	-1.03	1.45	-1.01	1.12	-1.01	2.20
C-RELOAD-14c	308	342	-1.00	1.53	-1.00	1.08	-1.00	2.01
C-RELOAD-14d	308	342	-1.03	1.48	-1.03	0.99	-1.03	2.17
C-RELOAD-14e	308	342	-1.03	1.54	-1.03	1.03	-1.03	2.28
C-RELOAD-q-24	968	632	-1.13	143.05	-1.13	56.34	-1.13	36.85
C-RELOAD-q-25	1033	658	-1.12	210.43	-1.12	121.46	-1.12	50.47
C-RELOAD-q-49	1430	3733	–		–	–	-1.13	69.45
C-RELOAD-q-104	3338	13936	–	–	–	–	-1.14	353.74
C-SCHED1	16	73	-3.06E4	0.42	-3.06E4	0.32	-3.06E4	0.64
Ex12.6.3	57	92	19.6	23	19.6	423.1	19.6	13.43
Ex12.6.4	57	88	8.6	70	8.6	478.2	8.6	2.94
Ex12.6.5	76	130	15.1	4	10.3	845.5	10.3	216.72
Ex12.6.6	97	180	16.3	18	16.3	937.4	16.3	149.40
FEEDLOC	247	89	0.0	145.49	0.0	252.41	0.0	157.39
MITTELMAN	7	16	16	0.26	16	0.01	16	0.01
OPTPRLOC	29	30	-8.06	0.87	-8.06	3.74	-8.06	1.58

continued on next page

continued from previous page

ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
PUMP	34	24	–	–	131124	977	130788	84.53
SPACE-25	235	893	484.33	183.54	484.33	321.01	484.33	190.42
SPACE-25-r	160	818	484.33	124.58	484.33	248.94	484.33	127.75
SPACE-960-i	6497	5537	–	–	–	–	7.65E6	187.43
SPACE-960	8417	15137	–	–	–	–	7.84E6	1206.43
SPACE-960-r	5537	12257	–	–	–	–	5.13E6	160.45
SPRING	8	17	0.86	0.08	0.86	0.74	0.86	0.59
STOCKCYCLE	97	480	–	–	436341	n/a	119948.7	6.45
TRIMLON2	12	8	5.3	3.42	5.3	4.11	5.3	4.39
TRIMLON4	24	24	12.2	10	8.3	11.0	8.3	2.73
TRIMLON5	30	35	12.5	14	10.3	55.3	10.3	24.5
TRIMLON6	36	48	18.8	19	15.6	1092.9	15.6	15.94
TRIMLON7	42	63	–	–	17.5	990.7	18.1	65.34
TRIMLON12	72	168	–	–	–	–	95.5	345.50
TRIMLOSS4	64	105	10.8	99	–	–	10.6	9.76
TRIMLOSS5	90	161	12.6	190	–	–	10.7	76.85
TRIMLOSS6	120	215	–	–	–	–	22.1	69.03
TRIMLOSS7	154	345	–	–	–	–	26.7	59.32
TRIMLOSS12	384	800	–	–	–	–	138.8	323.94
WIND-FAC	14	15	0.25	7.52	0.25	2.95	0.25	7.91

Table 5.5: Results on solving selected CNLP benchmarks from the CUTE library [13]. Results of all solvers were collected on an AMD Athlon MP2800 PC running RH Linux AS4 and a time limit of 3,600 sec. Solutions with the best quality are boxed. "–" means that no feasible solutions were found in the time limit.

Test Problem			LANCELOT		SNOPT		CPOPT(SNOPT)	
ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
DTOC6	5000	10001	-	-	-	-	1.02E6	58.05
EIGMAXB	101	101	0.91	1.34	-	-	1.87	24.33
GILBERT	1000	1000	2459.46	1.12	4700.61	689.18	2454.67	39.55
HADAMARD	256	129	-	-	-	-	0.99	7.88
KISSING	903	127	0.84	123.43	-	-	0.77	73.45
OPTCDEG	4000	6001	-	-	45.76	10.23	46.98	19.65
ORTHREGC	5000	10005	-	-	3469.05	557.98	2614.34	143.65
ORTHREGD	5000	10003	-	-	8729.64	208.27	7932.92	123.49
ORTHRGDM	5000	10003	1513.80	4.56	10167.82	250.00	2340.34	20.34
ORTHRGDS	5000	10003	912.41	4.20	-	-	894.65	105.34
VANDERM1	199	100	-	-	-	-	0.0	45.34
VANDERM3	199	100	-	-	-	-	0.0	36.70
VANDERM4	199	100	-	-	-	-	0.0	52.33

Table 5.7: Results on solving CNLP benchmarks from the CUTE library [13]. Results of all solvers were collected on an AMD Athlon MP2800 PC running RH Linux AS4 and a time limit of 3,600 sec. Solutions with the best quality are boxed. "−" means that no feasible solutions were found in the time limit.

Test Problem			LANCELOT		SNOPT		CPOPT(SNOPT)	
ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
ALJAZZAF	3	1	75.0	0.46	75.00	0.01	75.00	0.10
ALLINITC	4	1	30.44	*	30.49	0.01	30.49	0.10
ALSOTAME	2	1	0.082	0.57	0.08	0.01	0.08	0.09
AVION2	49	15	-	-	94680129.58	0.01	94680129.58	0.10
BATCH	46	73	-	-	259180.35	0.01	259180.35	0.11
BT11	5	3	0.825	0.62	0.82	0.01	0.82	0.09
BT12	5	3	6.188	0.47	6.19	0.01	6.19	0.09
BT6	5	2	0.277	0.56	0.28	0.01	0.28	0.09
BT7	5	3	306.5	0.51	360.38	0.01	360.38	0.09
BT8	5	2	1.0	0.57	1.00	0.01	1.00	0.09
CB2	3	3	1.952	0.60	1.95	0.01	1.95	0.09
CRESC4	6	8	-	-	0.87	0.01	0.87	0.10
CSFI1	5	4	-49.07	0.63	-49.08	0.01	-49.08	0.09
DIPIGRI	7	4	680.6	0.68	680.63	0.01	680.63	0.09
DIXCHLNG	10	5	0.0	1.12	2471.90	0.01	2471.90	0.10
DNIEPER	61	24	1.87×10^4	0.83	18744.01	0.01	18744.01	0.13
EXPFITA	5	22	1.13×10^{-3}	0.65	0.00	0.01	0.00	0.10
GAUSSELM	14	11	-2.25	0.55	0.00	104.90	0.00	0.12
HIMMELBI	100	12	-1735.6	1.23	-1755.00	0.01	-1755.00	0.13
HIMMELBJ	45	14	-	-	-1755.00	0.01	-1755.00	0.09

continued on next page

continued from previous page

ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
HIMMELP2	2	1	-62.05	0.63	-62.05	0.01	-62.05	0.09
HIMMELP6	2	5	-59.01	0.69	-59.01	0.01	-59.01	0.09
HONG	4	1	22.57	0.50	1.35	0.01	1.35	0.09
HS100	7	4	680.6	0.72	680.63	0.01	680.63	0.09
HS101	7	5	1809.7	*	1809.76	0.01	1809.76	0.14
HS102	7	5	911.9	*	911.88	0.01	911.88	0.12
HS103	7	5	-	-	543.67	0.01	543.67	0.11
HS104	8	5	3.95	0.58	3.95	0.01	3.95	0.10
HS107	9	6	5055	0.59	5055.01	0.01	5055.01	0.10
HS108	9	13	-0.866	0.58	0.00	0.01	0.00	0.10
HS109	9	10	-	-	0.00	0.01	0.00	0.10
HS111	10	3	-47.76	0.83	-47.71	0.01	-47.71	0.11
HS114	10	11	-1768.8	1.64	-1768.81	0.01	-1768.81	0.10
HS117	15	5	32.35	0.60	32.35	0.01	32.35	0.10
HS119	16	8	244.9	0.54	244.90	0.01	244.90	0.10
HS12	2	1	-30.0	0.46	-30.00	0.01	-30.00	0.09
HS18	2	2	5.0	0.65	5.00	0.01	5.00	0.10
HS19	2	2	-6961.8	0.58	-6961.81	0.01	-6961.81	0.09
HS20	2	3	40.2	0.52	38.20	0.01	38.20	0.09
HS23	2	5	2.0	0.54	2.0	0.01	2.0	0.09
HS24	2	3	-1.0	0.55	0.00	0.01	0.00	0.09
HS26	3	1	0.0	0.65	0.00	0.01	0.00	0.09
HS27	3	1	0.04	0.49	0.04	0.01	0.04	0.09
HS29	3	1	-22.6	0.53	0.00	0.01	0.00	0.09

continued on next page

continued from previous page

ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
HS30	3	1	1.0	0.52	1.00	0.01	1.00	0.09
HS32	3	2	1.0	0.54	1.00	0.01	1.00	0.09
HS33	3	2	-4.0	0.55	-4.0	0.01	-4.0	0.09
HS34	3	2	-0.834	0.38	-0.83	0.01	-0.83	0.09
HS36	3	1	-3300	0.55	0.00	0.01	0.00	0.09
HS37	3	2	-3456	0.48	0.00	0.01	0.00	0.10
HS39	4	2	-1.0	0.52	-1.00	0.01	-1.00	0.09
HS40	4	3	-0.25	0.58	-0.25	0.01	-0.25	0.09
HS41	4	1	1.926	0.52	2.00	0.01	2.00	0.09
HS42	4	2	13.86	0.56	13.86	0.01	13.86	0.09
HS43	4	3	-44.0	0.49	-44.00	0.01	-44.00	0.09
HS46	5	2	0.0	0.54	0.00	0.01	0.00	0.09
HS54	6	1	0.0	0.58	0.19	0.01	0.19	0.09
HS55	6	6	6.667	0.49	6.33	0.01	6.33	0.09
HS56	7	4	-3.456	0.55	0.00	0.01	0.00	0.12
HS57	2	1	0.03065	0.57	1.70	0.01	1.70	0.12
HS60	3	1	0.0326	0.62	0.03	0.01	0.03	0.09
HS61	3	2	-143.65	0.57	-143.65	0.01	-143.65	0.10
HS62	3	1	-26273	0.61	-26272.51	0.01	-26272.51	0.10
HS63	3	2	961.72	0.55	961.72	0.01	961.72	0.09
HS7	2	1	-1.732	0.56	-1.73	0.01	-1.73	0.09
HS71	4	2	17.01	0.62	17.01	0.01	17.01	0.09
HS73	4	3	29.9	0.52	29.89	0.01	29.89	0.10
HS74	4	5	5126.5	0.50	29.89	0.01	29.89	0.09

continued on next page

continued from previous page

ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
HS75	4	5	5174.4	0.56	29.89	0.01	29.89	0.09
HS77	5	2	0.2415	0.56	0.24	0.01	0.24	0.09
HS78	5	3	-2.92	0.58	-2.92	0.01	-2.92	0.09
HS79	5	3	0.0788	0.57	13.97	0.01	13.97	0.09
HS80	5	3	0.054	0.58	0.054	0.01	0.054	0.09
HS83	5	3	-30666	0.52	-30665.54	0.01	-30665.54	0.09
HUBFIT	2	1	0.0169	0.46	0.02	0.01	0.02	0.09
LOADBAL	31	31	0.453	0.69	0.45	0.01	0.45	0.12
LOOTSMA	3	2	-	-	2.00	0.01	2.00	0.09
MADSEN	3	6	0.616	0.55	0.62	0.01	0.62	0.09
MARATOS	2	1	-1.0	0.40	-1.00	0.01	-1.00	0.09
MATRIX2	6	2	0.0	0.52	0.00	0.01	0.00	0.09
MISTAKE	9	13	-1.0	0.58	-1.00	0.01	-1.00	0.10
MWRIGHT	5	3	24.97	0.56	24.98	0.01	24.98	0.10
NGONE	8	8	-0.5	0.51	24.98	0.15	24.98	0.27
ODFITS	10	6	-2380	0.50	-2380.03	0.01	-2380.03	0.10
OPTCNTRL	32	20	550	0.51	550.00	0.01	550.00	0.10
OPTPRLOC	30	30	-16.42	4.02	-16.42	0.01	-16.42	0.11
ORTHREGB	27	6	0.0	0.76	0.00	0.01	0.00	0.10
PENTAGON	6	15	1.509×10^{-4}	0.56	0.00	0.01	0.00	0.09
POLAK1	3	2	2.718	0.53	2.72	0.01	2.72	0.09
POLAK3	12	10	5.933	0.82	5.93	0.02	5.93	0.15
POLAK5	3	2	50.0	0.52	50.00	0.01	50.00	0.10
POLAK6	5	4	-44.0	0.74	-44.00	0.01	-44.00	0.11

continued on next page

continued from previous page

ID	n_c	n_v	Sol.	Time	Sol.	Time	Sol.	Time
RK23	17	11	0.0833	0.75	0.08	0.01	0.08	0.09
ROBOT	14	2	5.463	0.55	38.49	0.01	38.49	0.10
SINROSNB	2	1	0.0	0.56	38.49	0.01	38.49	0.11
SNAKE	2	2	-	-	-0.00	0.01	-0.00	0.11
SPIRAL	3	2	0.0	0.71	-0.00	0.01	-0.00	0.11
STANCMIN	3	2	4.25	0.58	4.25	0.01	4.25	0.09
SVANBERG	10	10	15.73	0.59	4.25	7.02	4.25	0.46
SYNTHES1	6	6	0.759	0.55	0.76	0.01	0.76	0.09
TWOBARS	2	2	1.51	0.53	1.51	0.01	1.51	0.09
WOMFLET	3	3	0.0	0.51	0.00	0.01	0.00	0.09
ZECEVIC3	2	2	97.31	0.54	97.31	0.01	97.31	0.09
ZECEVIC4	2	2	7.558	0.59	7.56	0.01	7.56	0.09
ZY2	3	2	2.0	0.46	2.00	0.01	2.00	0.09

5.4 Summary

In this section, we have presented a constraint-partitioning approach that exploits the constraint structure of large-scale MINLP and CNLP benchmark problems. We have developed an automated algorithm for partitioning a large problem in order to minimize the number of global constraints, an iterative method for determining the optimal number of partitions in order to minimize the search time, and an efficient strategy for resolving inconsistent global constraints based on the theory of extended saddle points. Our experimental results demonstrate significant improvements over the best existing solvers in terms of solution time and quality in solving a collection of mixed-integer and continuous nonlinear constrained optimization benchmarks.

Chapter 6

Conclusions and Future Work

In this chapter, we conclude our research on constraint partitioning using the theory of extended saddle points and points out some possible future directions.

6.1 Summary of Work

In this thesis, we have proposed a general approach of constraint partitioning to significantly reduce the computational complexity in solving constrained nonlinear optimization problems in discrete, continuous, and mixed spaces. This approach is based on the observations that most NLPs from real-world applications have structured constraints that are highly localized, and that exploiting the constraint structure by partitioning can lead to much relaxed subproblems that are easy to solve and are loosely coupled.

The constraint partitioning approach leads to global constraints that many be inconsistent across multiple subproblems. We have proposed a unified theory of extended saddle-point condition (ESPC) for solving discrete, continuous, and mixed NLPs. This work extends the previous work on ESPC for discrete NLPs, developed by Wah and Wu [91, 98], to a unified theory that works in discrete, continuous, and mixed spaces. Based on an ℓ_1-penalty formulation, our theory provides a necessary and sufficient condition that governs all constrained local minima. The result shows that each constrained local minimum is associated with a saddle point of the corresponding unconstrained ℓ_1-penalty function when penalties are sufficiently large.

In order to resolve violated global constraints efficiently, We further show that ESPC can be decomposed for constraint-partitioned NLPs. We derive a set of decomposed conditions that are necessary individually and sufficient collectively for constrained local minima. Since the decomposed ESPC is satisfied by constrained local minima in each subproblem in addition to satisfying the local constraints, it is much more effective than the local constraints alone for limiting the search space when resolving violated global constraints. Based on the decomposed ESPC, we propose a general partition-and-resolve framework that iterates between calling a basic solver to solve a modified subproblem while incorporating the local constraints and the bias on violated global constraints, and using a penalty-reweighting strategy to resolve the violated global constraints across the subproblems.

We have applied constraint partitioning to solve some planning and mathematical programming benchmarks originated from engineering applications. We have studied important implementation issues in making the constraint partitioning approach efficient for these applications, including automated methods in deciding partitioning dimensions, strategies in choosing the optimal number of partitions, selection and adaptation of existing solvers for solving subproblems efficiently, and strategies for updating penalty values in resolving violated global constraints quickly. For both applications, our proposed method has achieved significant reduction in solution time, and has solved many large problems not solvable by other start-of-the-art solvers.

6.2 Future Work

In this thesis, we have shown that the constraint partitioning approach can significantly reduce the search complexity by improving over existing solvers on planning and mathematical programming applications. We point out two possible ways to extend this research.

1) We plan to continue the study of the constraint partitioning approach for nonlinear optimization on other applications, including computational biology, VLSI circuit design, computer vision and optimal control. Many applications have intrinsic structural properties that can be exploited to make the search more efficient than general solution methods. Existing methods either do not utilize the structural information of applications (such as sequential quadratic programming), or require user-supplied domain-specific knowledge on problem structure (such as methods for circuit layouts). There is a lack of methods that automatically utilize problem structures, such as the structural relations between constraints and variables. It is desirable to design automated methods to exploit these problem structures using an application-oriented approach, and develop theoretical results on search complexity. Future research in this direction will improve the efficiency of nonlinear optimization methods and bridge the gap between optimization theory and practical applications.

2) The planning technology based on constraint partitioning has been proved very effective on the PDDL2.2 and the ASPEN models. We plan to study other important planning applications, such as spacecraft control and production planning, investigate the structure of these applications, and propose suitable schemes to exploit their structure to improve the search efficiency. Moreover, we plan to extend the proposed approach to more expressive AI modelling languages and to probabilistic planning and scheduling where there are uncertainties in the environments and outcomes of actions. In a broader scope, we plan to extend our planning technology and mathematical programming approach to other related subjects in artificial intelligence, including motion planning, robot planning, machine learning, and multi-agent systems.

References

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. J. Wiley and Sons, 1989.

[2] S. Anily and A. Federgruen. Ergodicity in parametric nonstationary Markov chains: An application to simulated annealing methods. *Operations Research*, 35(6):867–874, 1987.

[3] S. Anily and A Federgruen. Simulated annealing methods with general acceptance probabilities. *Journal of Appl. Prob.*, 24:657–667, 1987.

[4] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Prentice-Hall, Inc., Englewood Cliffs, NJ., 1976.

[5] T. Back, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9, 1991.

[6] T. Back, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 2–9, 1991.

[7] M. S. Bazaraa and J. J. Goode. A survey of various tactics for generating Lagrangian multipliers in the context of Lagrangian duality. *European Journal of Operational Research*, 3:322–338, 1979.

[8] J. C. Bean and A. B. Hadj-Alouane. A dual genetic algorithm for bounded integer programs. In *Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan*, 1992.

[9] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.

[10] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 1999.

[11] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[12] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence, Special issue on Heuristic Search*, 129(1), 2001.

[13] I. Bongartz, A. R. Conn, N. Gould, and P. L. Toint. CUTE: Constrained and unconstrained testing environment. *ACM Trans. on Mathematical Software*, 21(1):123–160, 1995.

[14] Y. X. Chen. *Optimal Anytime Search for Constrained Nonlinear Programming*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 2001.

[15] 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.

[16] S. Chien, *et al.* ASPEN - Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*. Space Operations Organization, 2000.

[17] A. R. Conn, N. Gould, and Ph. L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming*, 73:73–110, 1996.

[18] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programming. *Operations Research*, 8:101–111, 1960.

[19] P. Doherty and J. Kvarnstrm. TALplanner: An empirical investigation of a temporal logic-based forward chaining planner. In *Proc. Sixth Int'l Workshop on Temporal Logic-based Forward Chaining Planner*, pages 47–54. AIPS, 1999.

[20] M. A. Duran and I. E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:306–307, 1986.

[21] S. Edelkamp. Mixed propositional and numerical planning in the model checking integrated planning system. In *Proc. Workshop on Planning in Temporal Domains*, pages 47–52. AIPS, 2002.

[22] S. Edelkamp. Pddl2.2 planning in the model checking integrated environment. In *UK Planning and Scheduling Special Interest Group (PlanSig)*. Glasgow, 2003.

[23] S. Edelkamp and J. Hoffmann. Classical part, 4th international planning competition. http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/, 2004.

[24] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks Cole Publishing Company, 2002.

[25] M. P. Fourman. Propositional planning. In *Proc. Workshop on Model Theoretic Approaches to Planning*. AIPS, 2000.

[26] M. I. Freidlin and A. D. Wentzell. *Random perturbations of dynamical systems*. New York : Springer, 1984, 1984.

[27] B. Gavish. On obtaining the 'best' multilpliers for a Lagrangean relaxation for integer programming. *Comput. & Ops. Res.*, 5:55–71, 1978.

[28] A. M. Geoffrion. Generalized Benders decomposition. *J. Optim. Theory and Appl.*, 10(4):237–241, 1972.

[29] A. M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[30] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs with action costs. In *Proc. of the Sixth Int. Conf. on AI Planning and Scheduling*, pages 12–22. Morgan Kaufman, 2002.

[31] F. R. Giles and W. R. Pulleyblank. *Total Dual Integrality and Integer Polyhedra*, volume 25. Elsevier North Holland, Inc., 1979.

[32] P. E. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.

[33] F. Glover and G. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Proc. of Int'l Conf. on Metaheuristics for Optimization*, pages 113–133, 1995.

[34] N. I. M. Gould, D. Orban, and Ph. L. Toint. An interior-point ℓ_1-penalty method for nonlinear optimization. Technical report, RAL-TR-2003-022, Rutherford Appleton Laboratory Chilton, Oxfordshire, UK, 2003.

[35] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329, 1988.

[36] I. Harjunkoski, T. Westerlund, R. Pörn, and H. Skrifvars. Different transformations for solving non–convex trim loss problems by MINLP. *European Journal of Operations Research*, 105:594–603, 1998.

[37] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of Artificial Intelligence Research*, 14:253–302, 2001.

[38] K. Holmberg. On the convergence of the cross decomposition. *Mathematical Programming*, 47:269–316, 1990.

[39] K. Holmberg. Generalized cross decomposition applied to nonlinear integer programming problems: Duality gaps and convexification in parts. *Optimization*, 23:341–356, 1992.

[40] A. Homaifar, S. H-Y. Lai, and X. Qi. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–254, 1994.

[41] L. Ingber. *Adaptive Simulated Annealing (ASA)*. Lester Ingber Research, 1995.

[42] J. Joines and C. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. In *Proceedings of the First IEEE International Conference on Evolutionary Computation*, pages 579–584, 1994.

[43] A. E. W. Jones and G. W. Forbes. An adaptive simulated annealing algorithm for global optimization over continuous variables. *Journal of Optimization Theory and Applications*, 6:1–37, 1995.

[44] A. K. Jónsson, 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.

[45] A. K. Jónsson, P. H. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. National Conf. on Artificial Intelligence*. AAAI, 2000.

[46] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proc. 13th National Conference on Artificial Intelligence*, pages 1194–1201. AAAI, 1996.

[47] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. Int'l Joint Conf. on Artificial Intelligence*. IJCAI, 1999.

[48] H. Kautz and J. P. Walser. Integer optimization models of AI planning problems. *The Knowledge Engineering Review*, 15(1):101–117, 2000.

[49] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[50] J. Koehler and J. Hoffmann. On reasonable and forced goal ordering and their use in an agenda-driven planning algorithm. *J. of Artificial Intelligence Research*, 12:339–386, 2000.

[51] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proc. Second Berkeley Sympos. Math. Stat. Prob.*, pages 481–492. University of California Press, 1951.

[52] A. Kuri. A universal elctric genetic algorithm for constrained optimization. In *Proc. 6th European Congress on Intelligent Techniques and Soft Computing*, pages 518–522, 1998.

[53] L. S. Lasdon, A. D. Warren, A. Jain, and M. Ratner. Design and testing a generalized reduced gradient code for nonlinear programming. *ACM Trans. Math. Software*, 4:34–50, 1978.

[54] S. Leyffer. Mixed integer nonlinear programming solver. *http://www-unix.mcs.anl.gov/~leyffer/solvers.html*, 2002.

[55] S. Leyffer. MacMINLP: AMPL collection of MINLP problems. *http://www-unix.mcs.anl. gov/~leyffer/MacMINLP/*, 2003.

[56] F. Lin. A planner called R. *AI Magazine*, pages 73–76, 2001.

[57] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *J. of AI Research*, 10:87–115, 1999.

[58] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1984.

[59] Z. Michalewicz and C. Z. Janikow. Handling constraints in genetic algorithms. In *Proc. of 4th Int'l Conf. on Genetic Algorithms*, pages 151–157, 1991.

[60] D. Nau, H. Muoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 425–430. IJCAI, 2001.

[61] NEOS. The NEOS server for optimization. *http://www-neos.mcs.anl.gov/neos/*, 2005.

[62] R. S. Nigenda, X. Nguyen, and S. Kambhampati. AltAlt: Combining the advantages of Graphplan and heuristic state search. Technical report, Arizona State University, 2000.

[63] D. Orvosh and L. Davis. Shall we repair? genetic algorithms, combinatorial optimization, and feasibility constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, 1993.

[64] J. Penberethy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114. KR Inc., 1992.

[65] J. Penberethy and D. Weld. Temporal planning with continuous change. In *Proc. 12th National Conf. on AI*, pages 1010–1015. AAAI, 1994.

[66] J. Porteous, L. Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proc. European Conf. on Planning*, pages 37–48, 2001.

[67] D. Powell and M. M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proc. of 5th Int'l Conf. on Genetic Algorithms*, pages 424–431, 1993.

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

[69] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative repair planning for spacecraft operations in the ASPEN system. In *Proc. Int'l Symp. on Artificial Intelligence Robotics and Automation in Space*. European Space Agency, 1999.

[70] R. L. Rardin. *Optimization in Operations Research*. Prentice Hall, 1998.

[71] I. Refanidis and I. Vlahavas. The GRT planner. *AI Magazine*, pages 63–66, 2001.

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

[73] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proc. of 3rd Int'l Conf. on Genetic Algorithms*, pages 191–197, 1989.

[74] T. J. Van Roy. Cross decomposition for mixed integer programming. *Mathematical Programming*, 25:46–63, 1983.

[75] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–139, 1996.

[76] N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996.

[77] M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In *Proc. of 4th Parallel Problem Solving from Nature*, 1996.

[78] J. F. Shapiro. Generalized Lagrange multipliers in integer programming. *Operations Research*, 19:68–76, 1971.

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

[80] A. Tate, B. Drabble, and R. Kirby. O-Plan2: an open architecture for command, planning and control. *Intelligent Scheduling*, pages 213–239, 1994.

[81] J. Tind and L. A. Wolsey. An elementary survey of general duality theory in mathematical programming. *Mathematical Programming*, pages 241–261, 1981.

[82] A. Trouve. Rough large deviation estimates for the optimal convergence speed exponent of generalized simulated annealing algorithms. Technical report, LMENS-94-8, Ecole Normale Superieure, France, 1994.

[83] B. Wah and Y. X. Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, (accepted for publication) 2005.

[84] B. W. Wah and Y. X. Chen. Constrained genetic algorithms and their applications in nonlinear constrained optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 286–293. IEEE, November 2000.

[85] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. *Sixth Int'l Conf. on Principles and Practice of Constraint Programming*, pages 425–439, September 2000.

[86] B. W. Wah and Y. X. Chen. Hybrid constrained simulated annealing and genetic algorithms for nonlinear constrained optimization. In *Proc. IEEE Congress on Evolutionary Computation*, pages 925–932, May 2001.

[87] B. W. Wah and Y. X. Chen. Hybrid evolutionary and annealing algorithms for nonlinear discrete constrained optimization. *Int'l Journal of Computational Intelligence and Applications*, 3(4):331–355, December 2003.

[88] B. W. Wah and T. Wang. Constrained simulated annealing with applications in nonlinear constrained global optimization. In *Proc. Int'l Conf. on Tools with Artificial Intelligence*, pages 381–388. IEEE, November 1999.

[89] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 461–475. Springer-Verlag, October 1999.

[90] B. W. Wah and T. Wang. Tuning strategies in constrained simulated annealing for nonlinear global optimization. *Int'l J. of Artificial Intelligence Tools*, 9(1):3–25, March 2000.

[91] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 28–42. Springer-Verlag, October 1999.

[92] T. Wang. *Global Optimization for Constrained Nonlinear Programming*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, December 2000.

[93] T. Westerlund and F. Pettersson. A cutting plane method for solving convex minlp problems. *Computers and Chemical Engineering*, 19:S131–S136, 1995.

[94] D. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, pages 232–246, 1990.

[95] J. Willis, G. Rabideau, and C. Wilklow. The Citizen Explorer scheduling system. In *Proc. Aerospace Conf.* IEEE, 1999.

[96] S. Wolfman and D. Weld. Combining linear programming and satisfiability solving for resource planning. *The Knowledge Engineering Review*, 15(1), 2000.

[97] Z. Wu. *Discrete Lagrangian Methods for Solving Nonlinear Discrete Constrained Optimization Problems*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 1998.

[98] Z. Wu. *The Theory and Applications of Nonlinear Constrained Optimization using Lagrange Multipliers*. Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, May 2001.

[99] H. H. Zhang. *Improving Nonlinear Constrained Search Algorithms Through Constraint Relaxation*. M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, August 2001.

Vita

Yixin Chen received his B.S. degree from the Department of Computer Science, University of Science and Technology of China in 1999, and his M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign in May, 2001.

His research interests include nonlinear optimization, artificial intelligence, data warehousing, data mining, operations research, and algorithm complexity.