Hypergraph Partitioning for Exploiting Localities in Nonlinear Constrained Optimization*

Benjamin W. Wah and Soomin Lee
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
1308 West Main Street, Urbana, IL 61801, USA
URL: http://manip.crhc.uiuc.edu

Abstract

In this paper, we present a new hypergraph partitioning algorithm that jointly optimizes the number of hyperedge cuts and the number of shared vertices in nonlinear constrained optimization problems. By exploiting the localities of constraints with respect to their variables, we propose to partition the constraints into subproblems. We use a relaxed global search to solve the subproblems and resolve those violated global constraints across the subproblems by updating the corresponding penalties. As resolving violated global constraints is computationally expensive, we propose to reduce the number of global constraints by increasing the number of shared variables. This trade-off is advantageous because the number of global constraints in many benchmarks can be significantly reduced by having a small number of variables shared across the subproblems. Partitioning in this context can be achieved by partitioning the corresponding hypergraph. We improve hMETIS to jointly optimize the number of hyperedge cuts and the number of shared vertices. Our experimental results demonstrate improved solution quality with similar computational overhead on some VLSI cell placement benchmarks.

1 Introduction

Many applications in engineering, temporal planning, and VLSI design can be formulated as nonlinear programming problems (NLP) or mixed-integer nonlinear programming problems (MINLP). Such formulations have an *objective function* to be optimized and one or more *constraint functions* to be satisfied:

$$(P) \quad \begin{array}{ll} \min_{x,y} & f(x,y), \qquad x \in R^n, y \in Z^p \\ \text{subject to} & h(x,y) = 0 \text{ and } g(x,y) \leq 0, \end{array} \tag{1}$$

*Research supported by National Science Foundation Grant IIS 03-12084.

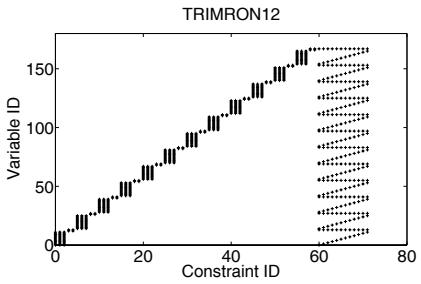


Figure 1. TRIMLON12 with strong localities.

where f and $h = (h_1, \ldots, h_m)^T$ and $g = (g_1, \ldots, g_r)^T$ are nonlinear functions. Our goal is to find an assignment of x and y that does not violate the constraints h and g and that minimizes the given objective function $f(x, y)$.

In the application problems studied in this research, we do not assume the continuity or differentiability of the objective and constraint functions. In fact, these functions do not have to be in closed form. The objective can be represented as a single function to be optimized or as multiple functions with trade-offs. The constraints may be *hard* and must be satisfied, or be *soft* and are optional.

We have observed that the constraints in many constrained optimization benchmarks have highly regular structures with respect to their variables. These *localities* happens because constraints are often constructed from related variables and are not generated randomly. For example, in a planning problem, constraints are used to model conditions with variables in close proximity in time. Such localities allow constraints to be clustered into multiple disjoint groups.

Figure 1 shows the constraint locality in TRIMLON12 [8] (with 168 mixed-integer variables and 72 constraints). The goal of the application modeled is to produce a set of paper-roll products from raw paper rolls by assigning some continuous and discrete variables in order to minimize a function of the trim loss and the production cost. Be-

cause the application involves 12 customers, the constraints can be clustered into 12 groups, each with 5 constraints. There are 60 local constraints (ID 0 to 59) that involve only local variables and 12 global constraints (ID 60 to 71) that involve variables from multiple groups.

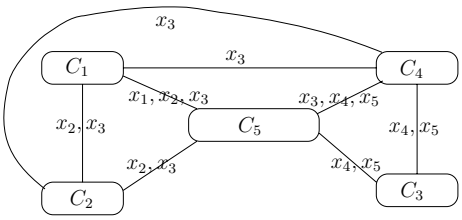
A lot of existing work on exploiting constraint localities has been done in the constraint-satisfaction-problem (CSP) community. (See for example the book by Dechter [4].) This belongs to a class of inferencing techniques that refine a constraint graph into a hypergraph or a hypertree, and that solve the problem using efficient constraint propagation and resolution strategies. Constraint resolution in CSP is based on the assumption that a constraint is defined either on discrete points or on a continuous interval when the constraint is linear. As a result, whether a constraint is satisfied can be verified by either enumeration or by more elegant constraint propagation and reasoning [4]. For example, the constraint $x \leq y$, where $x \in \{2, 3, 4\}$ and $y \in \{1, 2\}$, can be verified on each element in the Cartesian product of x and y.

In contrast, constraint localities are not effectively exploited in NLPs and MINLPs. The reason is that there are no effective techniques to verify whether a nonlinear constraint is satisfied in a continuous or mixed-integer domain, even for a single constraint. For example, there is no closed-form solution to find the zeroes of a polynomial equation in the continuous domain. General techniques are based on conditioning the instantiation of a subset of variables into a disjunction ($\vee$) of subproblems and on solving the subproblems one at a time until a solution is found. In cases when the domain is continuous and the functions are differentiable and satisfy some regularity conditions, the solution can be expressed as solving a system of nonlinear equations. More efficient procedures have been derived when the functions are convex [3]. Due to a lack of methods for resolving fine-grain nonlinear constraints, techniques in CSP for decomposing a constraint graph into hypergraphs or hypertrees do not lead to tractable solution procedures.

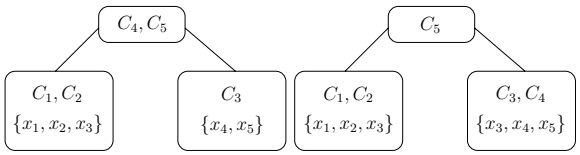
In our previous work [15], we have developed an effective approach for exploiting the localities of constraints in NLPs and MINLPs. Similar to that in CSPs, the constraints in an NLP or MINLP can be represented by a dual constraint hypergraph [4], where a node denotes a constraint and a hyperedge across multiple nodes denotes the variables shared by the corresponding constraints. Because of the difficulties in resolving individual nonlinear constraints, we only partition the constraints into clusters that are organized in two levels and develop algorithms for resolving each cluster of constraint as a whole. Each of the N lower-level clusters of constraints represents a *subproblem* of local constraints with local variables, and the subproblems are related by a cluster of global constraints that contain variables across multiple subproblems. Figure 2(a) shows an example of five constraints and their scope of variables, and Figure 2(b) shows the corresponding dual constraint graph.

$C_1(x_1, x_2, x_3)$
$C_2(x_2, x_3)$
$C_3(x_4, x_5)$
$C_4(x_3, x_4, x_5)$
$C_5(x_1, x_2, x_3, x_4, x_5)$

(a) Example constraints.



(b) The corresponding dual constraint graph [4].



(c) Constraint partitioning without shared variables.

(d) Constraint partitioning with variable x_3 shared.

Figure 2. Illustration of constraint partitioning

Based on our theory of extended saddle points [15], we have developed a *relaxed global search* (RGS) that solves each subproblem independently, while treating the top-level global constraints as soft constraints. Because we do not expect all the global constraints to be satisfied in each subproblem, we model their satisfaction as a penalty term in the objective function of a subproblem and minimize the weighted sum of those violated global constraints as well as the objective of the subproblem. Hence, RGS uses a modified objective function when solving Subproblem t:

$$\min_{x(t),y(t)} \quad f(x,y) + \gamma^T |H(x,y)| + \eta^T \max(0, G(x,y))$$
$$\text{subject to } h^{(t)}(x(t), y(t)) = 0 \text{ and } g^{(t)}(x(t), y(t)) \leq 0 \quad (2)$$

where $t = 1, \ldots, N$. Here, $h^{(t)}$ and $g^{(t)}$ are, respectively, the local equality and inequality constraint functions; $\{x(t), y(t)\}$ is a set of local variables involved in the local constraints; and $H(x, y)$ and $G(x, y)$ are, respectively, the equality and inequality global constraint functions.

With fixed γ and η, RGS solves each subproblem using an existing solver. After solving a subproblem, it increases γ and η on those violated constraints and repeat the process until either a feasible local minimum of (2) is found or when γ and η exceed their maximum bounds. Note that each penalty in γ and η is related to the duration that the corresponding constraint is violated. When a global constraint is not satisfied for a long time, its penalty becomes larger, and the solver pays more attention in satisfying it.

Our approach improves over a naive approach that solves

each subproblem independently without any consideration on satisfying the global constraints. Without the global constraints, the solution found by solving the subproblems alone will likely not satisfy the global constraints.

Note that RGS relies on an existing solver to resolve the nonlinear constraints in (2). By coordinating the solution process of the subproblems through the penalty term of the global constraints, RGS can resolve the global constraints very quickly, usually in a few iterations of solving the subproblems. Consequently, the complexity of RGS is significantly lower than that of existing NLP and MINLP solvers because each subproblem involves a small fraction of the constraints and is a significant relaxation of the original problem in complexity. It is possible to further reduce the complexity by decomposing the constraint hypergraph into more levels, although there will be more overhead in backtracking the resolution of the global constraints.

In RGS, the complexity for solving a problem includes the overhead for solving all the subproblems and that for resolving the violated global constraints. In some cases, when the number of global constraints is large, the overhead for consistently resolving all the global constraints across the subproblems can be computationally expensive [15, 16].

In this paper, we propose to use *shared variables* in the constraint partitioning process. We allow some variables to be shared across subproblems, rather than partitioning all the variables into disjoint sets. Based on the variables in a set, we group all those constraints that are related to the variables into a subproblem. As a result, constraints are disjoint across the subproblems, but variables may be shared. That is, from (2),

$$\begin{array}{ll} \text{Disjoint constraints:} & h^{(1)} \wedge h^{(2)} \wedge \ldots \wedge h^{(N)} = h \\ & g^{(1)} \wedge g^{(2)} \wedge \ldots \wedge g^{(N)} = g \\ \text{Shared variables:} & \bigcup_{t=1}^{N} x(t) = x. \end{array}$$

Our approach reduces the number of global constraints, since shared variables are more likely to cover some constraints that may otherwise be global. Figures 2(c) and 2(d) illustrate a smaller number of global constraints when x_3 is shared between the two subproblems.

Shared variables are not handled as global constraints in RGS. In maintaining the consistency of shared variables across subproblems, we propagate their values from those assigned in one subproblem to the next. The implicit global constraint corresponding to a shared variable does not carry any penalty and in that sense are weaker in arriving at consistent assignments. The aggregate complexity in this case is the total overhead for solving the subproblems, resolving the global constraints, and propagating the values of the shared variables. Although shared variables may incur new overheads in their propagation, they lead to good trade-offs in some benchmarks because a small increase in the number of shared variables may result in a significant decrease in the number of global constraints.

Our goal in this paper is to study the trade-offs between the number of shared variables and the number of global constraints. We achieve this goal by developing an efficient hypergraph partitioning algorithm that allows shared variables, based on the hypergraph partitioning package *hMETIS* [10]. We demonstrate the trade-offs on some NLP and MINLP benchmarks and illustrate the approach on solving the VLSI cell placement problem. By allowing shared cells in the cell placement problem, our results show reduced total wire lengths and similar CPU times when compared to the results of an existing placer Capo [1].

2 Hypergraph Partitioning

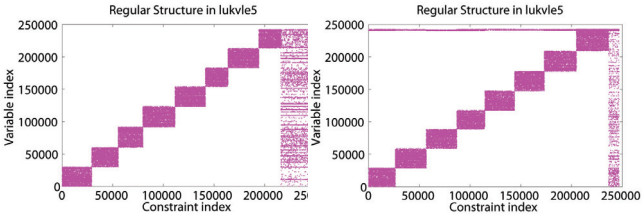
In this section, we survey some existing hypergraph partitioning algorithms for minimizing hyperedge cuts without shared variables. Based on a primal constraint graph, we associate the variables and constraints of an optimization problem to the vertices and hyperedges of a hypergraph, where each hyperedge connects a subset of the vertices. We further summarize the shortcomings of these algorithms with respect to our research. Last, we present two applications that are solvable by our approach.

2.1 Previous Work

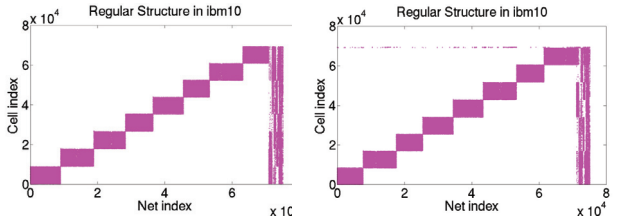
Hypergraph partitioning entails the process of finding disjoint partitions of the vertices or hyperedges in order to minimize some cost function. The problem of minimizing the number of hyperedges that span across multiple partitions is called the mincut problem. The decision problem *MINCUT_HG* [6] is defined as follows.

Definition 1. *MINCUT_HG.*
Input: *Hypergraph $G = (V, E)$, weight functions $w : V \rightarrow Z^+$ (a set of positive integers), $\ell : E \rightarrow Z^+$, and $J \in Z^+$ and $K \in Z^+$.*
Property: *There is a partition of V into disjoint subsets $V_1, V_2, \ldots, V_m$ such that $\sum_{e \in E'} \ell(e) \leq J$, where E' is a subset of the hyperedges whose elements belong to two or more partitions, under the balance constraints $\sum_{v \in V_i} w(v) \leq K$ for $1 \leq i \leq m$.*

The *Kernighan and Lin* (KL) method [12] and its improvement by Fiduccia and Mattheyses [5] iteratively improve in a pass a given partition by moving every vertex to its complementary partition exactly once. After all the moves have been made, the best partition encountered during the pass is taken as the output of the pass, and another pass begins with the best known partition. This class of methods are efficient and consistently produce good results. However, they needs dummy vertices for handling unbalanced problems, since they only produce exact bisections.



(a) lukvle5 without and with shared variables



(b) ibm10 without and with shared variables

Figure 3. The localized constraint structures in two constrained optimization benchmarks.

Hypergraph partitioning can be done by *hMETIS* [10], a software package that implements a randomized algorithm to partition a hypergraph $G = (V, E)$ into k subsets in such a way that bounds the number of vertices in each subset and that optimizes a partitioning objective. The algorithm can be applied to minimize either the number of shared variables without any global constraints or the number of global constraints without any shared variables. However, the algorithm cannot jointly optimize both objectives.

hMETIS [10] is an example implementation of this iterative improvement method. It produces M coarser graphs G_i by collapsing vertices and hyperedges of the original graph G_0, and by applying KL on the coarsest hypergraph $G_k = (V_M, E_M)$. It then refines the partition of each hypergraph G_i at level i by iteratively swapping vertices, using a similar criteria as in the partitioning process.

Simulated annealing (SA) [13] and *tabu search* (TS) [7] are well-known iterative heuristic algorithms based on random moves. SA randomly selects a vertex to be moved to another partition and always accepts such a move if it results in smaller hyperedge cuts and does not violate any given balance constraints. It also accepts worse moves with a probability that depends on the deterioration of the cost function and a control parameter called temperature. TS further avoids cycles in searches by taking advantage of the search history. Since SA and TS converge asymptotically, they may be able to achieve a better solution quality at the expense of an impractical amount of run time.

Spectral algorithms [2] find eigenvalues of the Laplacian matrix of the connectivity graph and derive a partitioning from the coefficients of an eigenvalue by comparing them to the median and by using constraints on their inputs. These algorithms are limited because they cannot handle fixed vertices well. As a result, they are not general for many applications, including VLSI placement and routing.

Network flow-based partitioning relies on the max-flow min-cut theorem and other efficient algorithms for identifying optimally small cuts in graphs [18]. The work by Hur and Lillis [9] applies similar techniques to VLSI placement using an incremental-flow solver. These polynomial-time algorithms do not typically perform approximation and can handle hyperedges using an accurate conversion to graphs. However, they cannot handle balance constraints, which results in expensive trial-and-error and slow run time.

Remarks. Existing algorithms can minimize either the number of shared variables without any global constraints or the number of global constraints without any shared variables, but not both jointly. To this end, we enhance *hMETIS* and study the trade-offs between the two objectives. By allowing shared variables, it is possible to significantly reduce the hyperedge cuts when compared to the original *hMETIS*.

2.2 Applications in Constraint Partitioning

We focus on benchmarks from two domains: NLP and VLSI design. These applications are very different in terms of their objective and constraint functions. Although domain-specific partitioning and resolution algorithms may work better for some of these benchmarks, our goal is to develop a general approach that is applicable without domain-specific tuning.

Nonlinear programming (NLP). The problem defined in (1) has ample applications in production management, optimal control and engineering designs. As is illustrated in Figure 1, NLP problems in real-world applications have useful localized constraint structures.

Figure 3(a) illustrates a strong constraint locality in the *lukvle5* NLP. Given 250,000 variables and 249,996 constraints, we need to find an assignment to the variables that optimizes the objective and that satisfies all the constraints. We can partition the constraints into 8 groups, resulting in either 31,241 global constraints without shared variables, or 10,464 global constraints with 2,063 shared variables.

VLSI placement and routing. Logic circuits are composed of gates (or standard cells) that are connected by metal wires. Using a common wire, the same electrical signal propagates through those connected components. Such a connection is called a net and can be represented by a hyperedge. The hypergraph corresponding to a logic circuit directly maps gates to vertices and nets to hyperedges.

In general, VLSI circuits have useful localized structures. A circuit can be grouped into sub-circuits according to their functionality, with only a few interconnections (such as clock and power lines) among the sub-circuits. We can produce partitions of a circuit with small inter-segment

communications by solving the corresponding hypergraph partitioning problem. In a top-down placement of large VLSI circuits, a region of a chip is divided geometrically, and each partitioned sub-circuit is placed in a subregion.

A VLSI placement and routing problem is as follows.

Definition 2. *VLSI Placement.*
Given hypergraph $G = (V, E)$, where $V = \{v_1, \ldots, v_N, v_{N+1}, \ldots, v_{N+P}\}$ is the set of cells, $E = \{e_1, \ldots, e_m\}$ is the set of nets (hyperedges), $e_i \subset V$, and $\{v_{N+1}, \ldots, v_{N+P}\}$ is the set of terminals with fixed locations throughout the placement process, find (x_j, y_j) of each cell v_j on the die without overlaps in order to minimize the sum of the wirelengths of all the nets $\sum_{i=1}^{m} WL(e_i)$.

Figure 3(b) illustrates a strong locality in *ibm10* (with 69,429 cells, 744 I/O terminals, 75,196 nets, and 297,567 pins) from the ISPD04 standard-cell benchmarks [14]. Given the I/O terminals on the perimeter, the problem entails the minimization of the total wirelength (the sum of wirelengths in each net) without overlap between cells. In general, a circuit can be grouped into sub-circuits according to their functionality, with only a few interconnections among them. In this example, we can partition the circuit into 8 groups, resulting in either 4,409 global nets without shared cells, or 4,054 global nets with 200 shared cells.

Note that the our hypergraph model does not model the hard overlap constraints of cells, but instead model each hyperedge as a net of connected cells. The reason is that the overlap constraints is enforced in the detailed placement phase of the placer. At the global level, it is more important to minimize the total wire length of the nets, while allowing cells that are shared across different partitions.

3 Proposed Approach

In this section, we formally define a new hypergraph partitioning problem with shared vertices and analyze its complexity. We also propose a new hypergraph partitioning algorithm that minimizes both hyperedge cuts and shared variables. Last, we present a VLSI design application that can be efficiently solved by our approach.

3.1 Problem Formulation and Complexity

As is discussed in Section 1, many real-world applications have highly localized constraint structures, and a partitioning of the constraints by their localities leads to subproblems with tightly coupled local constraints. Although the resulting subproblems are highly independent, there are still global constraints that span across the subproblems. These global constraints corresponds to hyperedge cuts and can be minimized by solving *MINCUT_HG* on the corresponding hypergraphs.

The localized structure and the number of global constraints are problem dependent, and there is a limit on how far they can be reduced by hypergraph partitioning alone. However, if we allow some variables to be shared among the partitions, then some global constraints will be covered by the shared local variables and become local constraints. We define this problem, *MINCUT_HG_SHARE*, as follows.

Definition 3. *MINCUT_HG_SHARE.*
Input: *Hypergraph $G = (V, E)$, weight functions $w : V \rightarrow Z^+$, $\ell : E \rightarrow Z^+$, and $J \in Z^+$ and $K \in Z^+$.*
Property: *There is a partition of V into subsets $V_1, V_2, \ldots, V_m$ with possibly $V_i \cap V_j \neq \emptyset$ for $i \neq j$ such that $\sum_{e \in E''} \ell(e) \leq J$, where E'' is a subset of hyperedges whose elements are strictly in two or more partitions, under the the balance constraints $\sum_{v \in V_i} w(v) \leq K$ for $1 \leq i \leq m$.*

Based on the NP-completeness of *MINCUT_HG* [6], we prove the NP-completeness of *MINCUT_HG_SHARE*.

Theorem 1. *MINCUT_HG_SHARE is NP-complete.*

Proof. We prove the NP-completeness by reducing *MINCUT_HG* to *MINCUT_HG_SHARE*.

It is obvious that *MINCUT_HG_SHARE* is in the class of NP: a nondeterministic algorithm can guess some subset of vertices with overlaps and check in polynomial time that the conditions in Definition 1 are met.

For *MINCUT_HG*, we are given a hypergraph $G = (V, E)$ with weight functions $w : V \rightarrow Z^+$ and $\ell : E \rightarrow Z^+$, balance constraint K, and constraint in the cut J. The *MINCUT_HG_SHARE* with an input $\tilde{G} = (\tilde{V}, \tilde{E})$, weight functions $\tilde{w} : \tilde{V} \rightarrow Z^+$, $\tilde{\ell} : \tilde{E} \rightarrow Z^+$, balance constraint $\tilde{K}$ and constraint in the cut $\tilde{J}$ can be reduced as follows:

$$\tilde{V}_i = V_i \cup \left\{ \bigcup_{j=1}^{m} V_{i,j}^* \right\} \qquad i = 1, \ldots, m, \tag{3}$$

$$\tilde{E} = E, \tag{4}$$

where $V_{i,j}^* \subseteq V_j,\ j = 1, \ldots, m$. Also,

$$\begin{aligned} \tilde{K} &= \sum_{v \in \tilde{V}_i} \tilde{w}(v) = \sum_{v \in V_i} w(v) + \sum_{\substack{v \in V_{i,j}^*, \\ i \neq j}} w(v) \\ &\leq K + (m-1)K = mK \end{aligned} \tag{5}$$

$$\begin{aligned} \tilde{J} &= \sum_{e \in E''} \tilde{\ell}(e) \\ &= \sum_{e \in E'} \ell(e) + \sum_{\substack{u \in V_{i,j}^*, i \neq j, \\ (u, u[k]) \notin E'}} \ell((u, u[k])) \leq J, \end{aligned} \tag{6}$$

where $u[k]$ are the adjacent vertices of u. This conversion is straightforward and can be done in polynomial time. □

1. **procedure** *HG_Partitioning_Share*
2. **for** $i = 0$ **to** $M - 1$
5. *coarsening(G_i);*
2. **end_for**
3. *partitioning(G_M)*;
2. **for** $i = M$ **to** 1
5. *refinement(G_i);*
6. *create_shared_vertices(G_i);*
7. *rollback(G_i);*
8. *propagation(G_i);*
2. **end_for**
10. **end_procedure**

Figure 4. New HG partitioning algorithm.

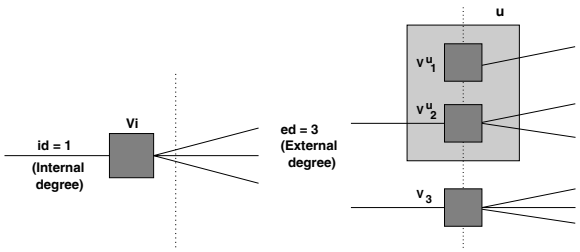
3.2 Overall Procedure

In this section, we develop an algorithm for solving *MINCUT_HG_SHARE* and study the trade-offs between the number of global constraints and the number of shared variables. By allowing shared variables, we can reduce the computational overhead for resolving the global constraints, while exploiting the inherent localized structures in constrained optimization problems.

Figure 4 presents our hypergraph partitioning procedure. The coarsening, partitioning and refinement phases are the same as those in the original multilevel approaches [11]. That is, in the coarsening phase, the original hypergraph $G_0 = (V_0, E_0)$ is transformed into a sequence of smaller graphs $G_1, G_2, \ldots, G_M$ such that $|V_0| > |V_1| > |V_2| > \cdots > |V_M|$. In the partitioning phase, a 2-way partitioning on the smallest graph G_M is computed in order to partition V_M into two parts, each containing half of the vertices of G_0. In the refinement phase, the partition of G_M is projected back to G_0 by going through intermediate graphs $G_{M-1}, G_{M-2}, \ldots, G_1, G_0$.

$HG\_Partitioning\_Share$ is implemented after the refinement phase. We first assign some vertices to be shared in a greedy fashion such that they maximally reduce the hyperedge cuts of the current partition. We define the *external degree* of vertex v as the number of hyperedges that contain v in one partition and its neighbor in another. This represents the actual reduction in the hyperedge cuts if v is shared. We also define the *internal degree* of v as the number of hyperedges that contain v and its neighbor in the same partition (Figure 5(a)). We put all the vertices in a priority queue according to their external degrees and retrieve one vertex at a time in order to make both partitions have that vertex in common.

Next, in the rollback computation, we restore some unnecessary shared vertices propagated from a lower-level graph to their original partitions. As is illustrated in Figure 5(b), u is a shared vertex from the lower-level graph G_{i+1}; v_1^u and v_2^u are shared vertices in the current-level graph G_i propagated from G_{i+1}; and v_3 is a shared vertex that is newly created in this level. Since v_1^u needs not be shared any more, namely, it has a zero external degree, we restore it to its original partition.



(a) Internal and external degrees (b) Roll back

Figure 5. Illustration of the algorithm.

Third, in the propagation phase, we project the shared vertices created in the current-level graph G_i to the upper-level graph G_{i-1}.

3.3 Application: VLSI Placement

We illustrate the application of *HG_Partitioning_Share* in solving the VLSI cell-placement problem.

We first map the given cells and nets into vertices and hyperedges. The hypergraph created only describes the logical connection of nets and does not convey any information about their layout and fixed I/O terminal interconnections. Next, we assign the fixed terminals to partitions according to their locations before applying our algorithm.

One of the most important considerations in a partitioning-based placement approach is to coordinate the result of each subproblem in such a way that its optimization is consistent with the optimization of the original problem. Since hypergraph partitioning leads to some hyperedge cuts (global nets) that have to be considered in multiple partitions, the optimization of these global nets requires an additional overhead in solving each subproblem. Our hypergraph partitioning algorithm minimizes this overhead by controlling the number of shared cells and the corresponding number of global nets.

Figure 6 presents our proposed partition-and-place procedure. It first partitions a subregion S (a whole die in the first iteration) into S_1 and S_2 by choosing a horizontal or vertical cutline with respect to the constraint on aspect ratio. Given the partitioned subproblems $P_i, i = 1, 2$, it finds the placement of P_i for subregion S_i using an existing placer until a stopping condition is met.

After partitioning, we have some shared cells C_s that belong to both partitions and private cells C_1 and C_2 that belong to, respectively, partitions 1 and 2. To allow P_i to be solved by an existing placer, we need an additional placement step P_s for shared cells. We first place C_s on a given subregion S and divide C_s into C_{s1} and C_{s2} according to their locations and the cutline. In solving P_1 (*resp.*, P_2), we

1. **procedure** *partition_and_place*(S)
2. **call** *partition()*; // partition cells and choose a cutline of subregion S //
3. **repeat** // outer loop //
4. **for** $i = 1$ **to** 2
5. *place*(P_i,S_i); // apply an existing placer //
6. **end_for**
7. *place*(P_s,S);
8. **until** stopping condition is satisfied
9. **end_procedure**

Figure 6. The partition-and-place framework.

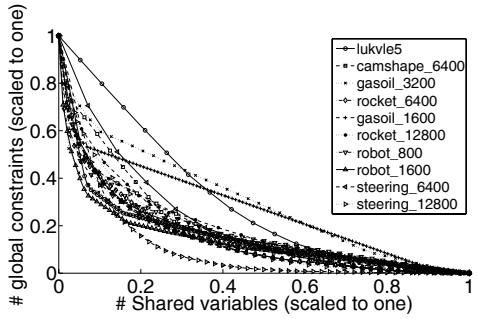
place C_1 and C_{s1} (*resp.*, C_2 and C_{s2}) in S_1 (*resp.*, S_2). In this step, we calculate both the wirelength of the local nets and that of the global nets.

4 Experimental Results

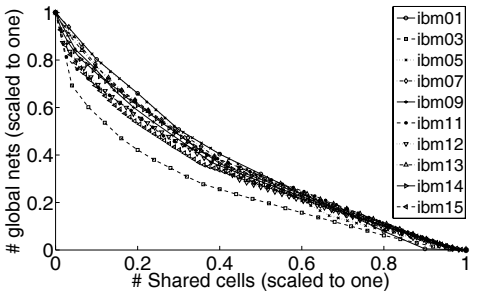
We have performed extensive experiments on our proposed algorithm using several NLP and VLSI placement benchmarks. Figure 7 shows the trade-offs between the number of global constraints and that of shared variables. We observe that all the benchmarks have similar and consistent trade-offs. For instance, the NLP benchmark *lukve5* shows 70% reduction in the number of global constraints with only 5% increase in the number of shared variables.

To illustrate the effects of shared variables and global constraints when solving a benchmark problem, we have applied the existing placer Capo [1] for finding the wirelengths and placements of the partitioned subproblems for those benchmarks in the ISPD04 [14] and ISPD06 [17] standard-cell suites. Our experiments were performed on an AMD 2-GHz computer running RedHat Linux 3.4.6 with 2 GB of main memory.

Table 1 summarizes our results by comparing the wirelengths of the partitions without shared variables (Ours w/o Share) and those of the partitions with shared variables (Ours w/ Share), when normalized with respect to the wirelengths found by the original Capo (run in the default mode). It also shows the normalized run times for finding those partitions. The results show, for a range of small and large benchmarks, that our placement algorithm with shared variables has about 8%-9% better average wirelengths when compared to, respectively, our algorithm without shared variables and Capo, with only 6% increase in run time. They demonstrate that incorporating shared variables in hypergraph partitioning can improve the solutions in partitioning-based constrained optimization.



(a) Trade-offs on some NLP Benchmarks



(b) Trade-offs on some ISPD04 benchmarks

Figure 7. Trade-offs between the number of global constraints and that of shared variables.

References

[1] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov. Unification of partitioning, placement and floorplanning. In *Proc. IEEE/ACM Int'l Conf. on Computer-aided Design*, pages 550–557, 2004.

[2] C. J. Alpert and A. B. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. In *Design Automation Conf.*, pages 652–657, 1994.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univesity Press, 2004.

[4] R. Dechter. *Constraint Processing*. Morgan Kaufman, 2003.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Conf. on Design Automation*, pages 175–181, Piscataway, NJ, 1982. IEEE Press.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1979.

[7] F. Glover, E. Taillard, and D. Werra. A user's guide to tabu search. *Ann. Oper. Res.*, 41(1-4):3–28, 1993.

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

[9] S. Hur and J. Lillis. Mongrel: Hybrid techniques for standard cell placement. *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design*, page 165, 2000.

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Applications in VLSI domain. Technical report, University of Minnesota, 1997.

Table 1. A comparison of normalized wirelengths and run times on the ISPD2004 [14] and ISPD2006 [17] standard-cell benchmark suites.

Circuit	Nodes	Term'ls	Nets	Pins	Capo		Ours w/o Shar		Ours w/ Shar	
					WL	Run time	WL	Run time	WL	Run time
ibm01	12752	246	14111	50566	1821320	143.43	1.00	1.01	0.90	1.10
ibm02	19601	259	19583	81199	3901240	305.07	0.98	1.03	0.92	1.08
ibm03	23136	283	27401	93573	4922910	372.46	1.00	1.04	0.95	1.06
ibm04	27507	287	31970	105859	6142910	510.31	0.98	0.99	0.94	1.00
ibm05	29347	1201	28446	126308	10051300	597.20	1.01	0.99	0.94	1.04
ibm06	32498	166	34826	128182	5678200	525.31	0.96	1.00	0.87	1.09
ibm07	45926	287	48117	175639	9153920	831.97	1.00	1.01	0.91	1.08
ibm08	51309	286	50513	204890	9845100	983.71	0.97	0.99	0.91	1.02
ibm09	53395	285	60902	222088	10188800	1049.94	0.97	0.98	0.92	1.08
ibm10	69429	744	75196	297567	19025400	1264.47	1.01	1.02	0.93	1.11
ibm11	70558	406	81454	280786	15065200	1190.68	0.98	1.01	0.92	1.07
ibm12	71076	637	77240	317760	24282700	1443.38	0.98	1.00	0.92	1.07
ibm13	84199	490	99666	357075	18553100	1549.77	0.99	1.00	0.93	1.06
ibm14	147605	517	152772	546816	34769800	2783.13	0.97	1.01	0.93	1.10
ibm15	161570	383	186608	715823	41409300	3553.24	1.01	1.00	0.95	1.06
ibm16	183484	504	190048	778823	47531500	3820.03	1.00	1.02	0.91	1.13
ibm17	185495	743	189581	860036	67835800	4524.73	1.03	1.00	0.90	1.09
ibm18	210613	272	201920	819697	44153100	3757.06	0.98	1.02	0.91	1.09
adaptec5	843128	646	867798	3493147	491602333	9718.32	1.01	1.01	0.91	1.02
newblue1	330474	337	338901	1244342	98353111	2562.30	0.96	0.97	0.93	0.99
newblue2	441516	1277	465219	1773855	308639298	5641.50	0.96	1.01	0.89	1.05
newblue3	494011	11178	552199	1929892	361211449	6076.30	0.99	1.02	0.91	1.12
newblue4	646139	3422	637051	2499178	358281708	6926.42	0.98	1.08	0.92	1.14
newblue5	1233058	4881	1284251	4957843	657403089	20854.45	1.01	0.98	0.93	0.97
newblue6	1255039	6889	1288443	5307594	668332084	18484.68	0.99	1.03	0.94	1.01
newblue7	2507954	26582	2636820	10104920	1518490039	54962.42	1.01	1.02	0.95	1.02
Norm. Avg.	–	–	–	–	–	–	0.99	1.01	0.92	1.06

[11] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, University of Minnesota, 1995.

[12] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Technical Report V-29, The Bell System Laboratory, 1970.

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

[14] N. Viswanathan and C. Chu. Fastplace: Efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Int'l Symp. on Physical Design*, pages 26–33, 2004.

[15] B. Wah and Y. X. Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, 170(3):187–231, 2006.

[16] B. W. Wah and Y. X. Chen. Solving large-scale nonlinear programming problems by constraint partitioning. In *Proc. Principles and Practice of Constraint Programming, LCNS-3709*, pages 697–711. Springer-Verlag, Oct. 2005.

[17] Web Site for ISPD 2006. International symposium on physical design. `http://www.sigda.org/ispd2006/contest.html`, 2006.

[18] H. Yang and D. F. Wong. Efficient network flow based min-cut balanced partitioning. In *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design*, pages 50–55, Los Alamitos, CA, 1994. IEEE Computer Society Press.