Accepted for publication in Proc. Int'l Conf. on CAD (ICCAD), 2008

Algorithms for Simultaneous Consideration of Multiple Physical Synthesis Transforms for Timing Closure *

Huan Ren and Shantanu Dutt
Dept. of ECE, University of Illinois-Chicago

Abstract

We propose a post-placement physical synthesis algorithm that can apply multiple circuit synthesis and placement transforms on a placed circuit to improve the critical path delay under area constraints by simultaneously considering the benefits and costs of all transforms (as opposed to considering them sequentially after applying each transform). The circuit transforms we employ include, but are not limited to, incremental placement, two types of buffer insertion, cell resizing and cell replication. The problem is modeled as a min-cost network flow problem, in which nodes represent circuit transform options. By carefully determining the structure of the network graph and the cost of each arc, a set of near-optimal transform options can be obtained as those whose corresponding nodes in the network graph have the min-cost flow passing through them. We also tie the transform selection network graph to a detailed placement network graph with TD arc costs for cell movements. This enables our algorithms to incorporate considerations of detailed placement cost for each synthesis transform along with the basic cost of applying the transform in the circuit. We have tested our algorithms on three sets of benchmarks under 3-10% area increase constraints, and obtained up to 48% and an average of 27.8% timing improvement. Our average improvement is relatively 40% better (8.2% better by an absolute measure) than applying the same set of transforms in a good sequential order that is used in many current techniques. Considering only synthesis transforms (no replacement), our technique is relatively 50% better than the sequential approach.

1 Introduction

As IC designers keep pursuing high performance of integrated circuits, the regular design flow usually cannot satisfy the given timing requirement in one run. This is especially due to the inaccuracy of wire load estimation at high levels of the design flow (e.g., logic synthesis, technology mapping). Thus, post-placement physical synthesis methods for improving timing have attracted significant attention for achieving design closure. Since these methods are applied in the post-placement stage, they have sufficiently accurate interconnect timing information. Some of the popular methods are the following.

Timing-driven incremental placement [15, 5]. This method tries to reduce the critical and near critical path lengths by changing the position of cells on these paths. The advantage of incremental placement is that by only focusing on cells connected to critical and near-critical paths, the run time is greatly reduced compared to performing a new placement, and the optimization is more controllable.

Cell re-sizing. Changing input capacitances and driving resistances of cells are very effective in improving timing. If we are able to implement cells of any size, then the problem of choosing optimal cell sizes for timing can be solved by a convex programming approach [6]. However, since in real standard cell designs, the available cell sizes in a library is limited, the problem is actually a discrete optimization problem, which can be solved by either a fastest descent method [3] or dynamic programming [7].

Buffer insertion. Optimal buffer insertion has to take routing information into consideration [14]. However, it is useful to estimate the effect of buffers at the placement stage so that proper white-spaces can be allocated for adding buffers, and we can know earlier whether the design can meet timing requirements.

Cell replication [13]. In cell replication, the driving cell of a net is replaced by two identical replicas, and the fanouts are partitioned into two groups for each replica. Its effect is similar to doubling the driving cell size, but it can also achieve the purpose of separating the non-critical fanouts of a net from its critical ones by connecting them to different replicas. This makes it more effective than up-sizing driving cells when a net has large non-critical load.

Though each of the above circuit transforms has been extensively studied on its own, the problem of applying them optimally together has not been solved satisfactorily. Most combined algorithms simply apply them in sequential order [9, 11]. However, it has been proven in [8] that applying these methods sequentially will produce fairly non-optimal solutions.

A few efforts have been made to combine them more effectively. In [4], the placement transform is applied, to some degree, simultaneously with other synthesis transforms. This is done by incorporating synthesis transforms into different intermediate partition levels of a partition based placement process according to the amount of their potential perturbation to the placement. In [8], both buffer insertion and resizing choices are available for each net. A greedy method is used in which the transform with the largest delay improvement to area increase ratio is chosen. Though, these two methods give better results than the sequential approach, they are still liable to be trapped at local optima, and furthermore, are only able to combine a few predetermined transforms (in contrast to a general method, like the one developed in this paper, that is applicable to almost any given set of transforms).

In this paper, we propose a network flow based method that can simultaneously consider almost any set of transforms for determining a near-optimal selection of transforms. We instantiate our method for the previously mentioned five circuit transforms: incremental placement, cell re-sizing, buffer insertions of two types (see Fig. 1) and replication. In our method, the different transform options are modeled as nodes in the network graph. By solving a min-cost flow problem in the graph, in which the nodes with flow passing through them represent the selection of the corresponding options, we obtain a near-optimal timing solution——the near-optimality comes from having to constraint the flow to adhere to various discrete requirements like going through exactly one option node for each transform of a cell.

The contribution of this paper is mainly three folds. (1) We develop network graph structures as well as an accurate arc cost formulation, that is based on the objective function and the given transform set, for obtaining near-optimal physical synthesis solutions. (2) We propose techniques for satisfying the discrete requirements of the underlying transform application problem while using a continuous network flow based model. We ensure these requirements by attaching a carefully determined objective-function independent cost to arcs in the network graph. (3) We incorporate the detailed placement with the transform selection process. Our transform selection network graph (TSG) ties into a detailed placement network graph (DPG) developed in [5], such that the selected transforms, e.g., added buffers and replaced cells are automatically placed and legalized with the cost for the latter accounted for in determining the synthesis solution. To the best of our knowledge, no previous work has considered the effect of detailed placement on the objective function during physical synthesis.

The rest of the paper is organized as follows. Section 2 introduces our problem formulation. The overview of our method is provided in Sec. 3. In Sec. 4, we describe various issues in the basic building blocks of the TSG including techniques for accurately capturing

*This work was supported by NSF grant CCR-0204097.

the objective function value in the arc costs. Methods to satisfy the discrete requirement of consistent transform option selection are discussed in Sec. 5. In Sec. 6 we show how to determine the flow amount into each net structure, and thereby determine arc capacities. The coupling between the TSG and the DPG is explained in Sec. 7. Section 8 presents experimental results and we conclude in Sec. 9.

2 Problem Formulation

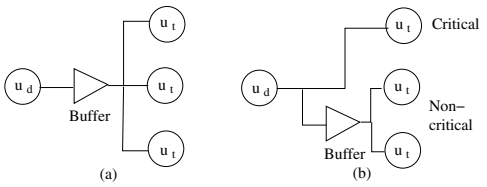

Figure 1: *Two types of buffers. u_d is the source cell and u_t is the sink cell of a net. (a) Type 1 buffer for improving the drive capability of u_d. (b) Type 2 buffer for isolating non-critical sinks from critical ones.*

Given a placed circuit $\mathcal{PC}$, we are allowed to resize, re-place, replicate cells in it and insert buffers. The objective is to produce a new placed circuit with improved critical path delay under the constraint that the total cell area increase is less than a given upper bound. We define $\mathcal{P}_\alpha$ to be the set of paths with a delay greater than a $1-\alpha$ ($\alpha < 1$) fraction of the most critical path delay. In our experiments, α is set to be 0.1. In order to reduce the complexity of the problem we only consider applying transforms on cells that are in $\mathcal{P}_\alpha$. This simplification does not limit the optimization potential of our method, since our method is incremental in its nature, and thus we can iterate it several times to take more paths into consideration until a desired circuit performance under the given constraints is reached (or failing which, the best possible performance under the given constraints is obtained).

We formulate this problem as an *option selection problem*. For each transform there is a set of options, and the goal is to choose one option from each transform option set (including choosing not to apply the transform) so that the critical path delay is minimized. Five transforms are considered in this paper as described below; note though that our method applies to any set of transforms.

(1) *Cell resizing*. The options are different sized cells provided by the library.

(2-3) *Buffer insertion*. As in [8], we consider two types of buffer insertions for each net with different goals. As shown in Fig. 1, a type 1 buffer is for improving the drive capability of the drive cell, while a type 2 buffer is for isolating fanouts in $\mathcal{P}_\alpha$ from other non-critical fanouts. The options for type 1 and type 2 buffer insertions are the different sized buffers in the library as well as not applying the buffering transform. Note that these two transforms are the only ones associated with nets and not with cells.

(4) *Cell replication*. The options for this transform includes no replication and replications with different partition schemes of the fanouts between the two replicas. However, if we provide all possible partitions as options, the number of options will be huge, since we need to consider the possibility that some fanout gates (sinks) may also be replicated. Hence, in our method, we pre-calculate the three best timing-driven partitions based on the initial $\mathcal{PC}$ using the algorithm in [13], and incorporate these as options. We thus have a total of four replication options for each cell: three replication options corresponding to the three chosen partitions and the no-replication option.

(5) *Replacement*. We pre-calculate the timing optimal positions of cells in $\mathcal{P}_\alpha$ with incremental timing-driven global placement. We then set up two options for cell replacement: remaining in the original position or being moved to the new position. It should be noted that moving cells to a global timing optimal but illegal placement position does not guarantee a timing improvement considering the detailed placement cost (which our algorithm does); hence the original position can be a better choice in some cases.

3 Overview of Our Network Flow Structure

We propose a network flow based method to solve the option selection problem. A transform selection network graph (TSG) is constructed for the circuit we consider. The different transform options are modeled as nodes (called *option nodes*) in the TSG. We use flow to select options. If there is flow passing through a node, the option corresponding to the node is selected. A simple example of the TSG corresponding to a two-net circuit is shown in Fig. 2. In the figure, $O_x(u)$, termed the *option set*, denotes the set of options of transform T_x for cell u. Two option nodes are shown for each option set. S is the source of the network, and T is the sink. The TSG in this example is a linear multilevel connection among option sets, in which option sets of each cell are sequentially arranged. Further, adjacent option sets in the entire structure are connected by complete bipartite subgraphs. The flow f in the figure indicates options $O_1^1(u)$, $O_1^2(u)$, ..., $O_2^2(w)$ are chosen, where $O_x^i(c)$ is the i'th option for transform T_x on a cell c. A flow that follows a different path through the TSG, e.g., f', represents a different combination of chosen transform options. Since adjacent sets of transform options are connected by a complete bipartite subgraph, any combination of transform options can be represented by some flow in the TSG.

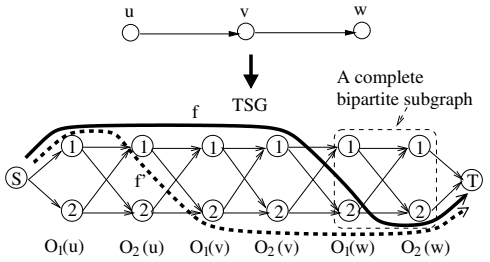

Figure 2: *A possible TSG for a two-net path.*

The *min-cost* flow problem is a classic network flow problem, which, given a cost of each arc, determines a minimum cost way of sending a certain amount of flow from the source (S) to the sink (T) in a network graph. If we can set the arc cost in such a way that for any flow in the TSG the total incurred cost of the flow is equal to the circuit delay objective function when transforms chosen by the flow are applied (we describe such an arc cost formulation in Sec. 4.2), and if the TSG provides all possible combinations of transform options (as is the case in the above multilevel linear TSG, and as is also the case in the more complex final TSG we describe in Sec. 4.2), then determining a min-cost flow in the network graph also determines the optimal transform option selection for the circuit. We also note that min-cost network flow algorithms (e.g., the simplex algorithm of [1]) do not enumerate all possible flow possibilities in reaching an optimal solution, but do so much more efficiently. Hence, our overall option selection algorithm FlowSynth that uses network flow also finds the optimal (or near-optimal, as it turns out in our case due to other factors to be described shortly) multi-transform solution much more efficiently than enumerating all possible transform option combinations.

3.1 The Timing Objective Function

One problem of using the min-cost flow to minimize circuit delay is that the min-cost flow just minimizes the summation of all costs, while minimizing the circuit delay requires minimization of the max path delay. The same problem is present in any timing optimization problem such as timing-driven placement, and a common remedy is to minimize a continuous timing objective function (instead of a discontinuous one like max) that is strongly correlated to the max function, i.e., one in which paths with higher delays have significantly higher contributions to the function than lower-delay paths. Thus by minimizing such a function, we will be minimizing essentially the highest delay paths, which is a good approximation of minimizing the max-delay path.

We use the timing objective function proposed in [5] that meets the above criterion. It is derived from an accurate pre-routed Elmore delay model. For a net n_j with driver u_d, let R_d be the driving resistance, C_i the input capacitance of a load cell u_i, C_g^i the total input capacitance of all the other load cells, r (c) the unit wire resistance (capacitance), $l_{d,i}$ the interconnect length connecting driver u_d to sink u_i and L_j be the total WL. Furthermore, let γ be the fraction of the wire length of n_j that is used by the interconnect to u_i. The delay $D(u_i, n_j)$ to a sink

u_i from the driver u_d, consists of three parts:

$$D_1(n_j) = R_d(c \cdot L_j + C_g^i + C_i) \quad (1)$$

$$D_2(u_i,n_j) = \frac{rc}{2} \cdot l_{d,i}^2 + r \cdot l_{d,i} C_i \quad (2)$$

$$D_3(u_i,n_j) = r \cdot (l_{d,i}/2)(1-\gamma/2)(c \cdot L_j + C_g^i) \quad (3)$$

$$D(u_i,n_j) = D_1(n_j) + D_2(u_i,n_j) + D_3(u_i,n_j) \quad (4)$$

The $D_1(n_j)$ delay component is common to all sinks of n_j, $D_2(u_i,n_j)$ is the wire RC delay from driver to a sink u_i, and $D_3(u_i,n_j)$ is the estimated delay of one WL section between u_d and u_i on the main trunk of the interconnect charging the rest of the interconnect and load capacitance of the net.

We define $CS(n_j)$ to be the set of critical sinks of n_j, which is: 1) all sinks in $\mathcal{P}_\alpha$ if the net is in $\mathcal{P}_\alpha$, or 2) the sink with the minimum slack otherwise. We then have the following delay objective functions F_t similar to the one in [5]:

$$F_t = \sum_{n_j} \sum_{u_i \in CS(n_j)} D(u_i,n_j)/S_a^\alpha(n_j) \quad (5)$$

where $S_a(n_j)$ is the allocated slack of n_j (slack of the max-delay path through n_j divided by the number of nets in the path), and α is the exponent of $S_a(n_j)$ used to adjust the weight difference in F_t between critical and non-critical paths. Based on experimental results, α is chosen as 1. By only considering critical sinks in F_t, we reduce the effect of fairly non-critical paths when performing timing optimization.

3.2 General Structure of the TSG

The structure of our TSG is more involved compared to the simple example given in Fig. 2. In order to handle complex circuits, instead of building the TSG for the entire circuit directly, we construct it hierarchically by first building a "mini TSG" for each net in P_α called a *net structure*. Then, net structures of connected nets are connected by *net spanning structures* to form the complete TSG as shown in Fig. 3. Thus, the TSG has a topology similar to the net interconnections in P_α. A net structure N_j is a *child net structure* of N_i if they are connected and N_j follows N_i in the flow direction in the TSG (i.e., the signal direction in the circuit is from net n_i to net n_j); correspondingly N_i is a *parent net structure* of N_j.

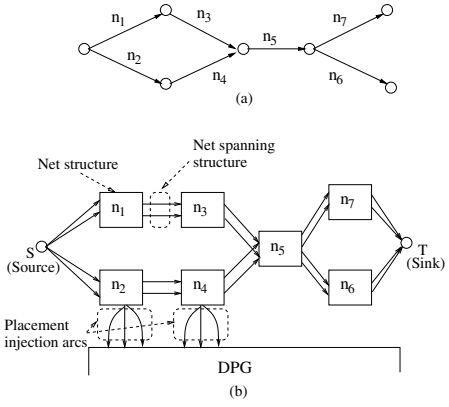

Figure 3: *(a) Nets in P_α of a circuit. (b) The corresponding TSG, which includes net structures, net spanning structures and placement injection arcs.*

A major characteristic of our method is that we incorporate the detailed placement graph (DPG) proposed in [5], which performs detailed placement with minimum timing cost, seamlessly with our TSG. Each net structure will send a flow with an amount equal to the total cell area of the chosen synthesis options in the net to the corresponding cell positions in the DPG through *placement injection* arcs. In this way, by obtaining the min-cost flow in the combined graph, we can achieve both transform selection and detailed placement of the resulting circuit in one shot, and the cost of the flow accounts for both steps. This property is novel and valuable since almost all the other transform algorithms perform the two steps separately, and thus are oblivious to the detailed placement cost on the objective function when performing physical synthesis.

A few major issues need to be tackled in order to obtain a correct, accurate and effective physical synthesis methodology using a min-cost network flow approach. These are:

Algorithm FlowSynth

1. Based on the given optimization objective function F and the given set $\mathcal{T}$ of transforms, construct a net structure (formed of overlapping *star-graphs*) for each net in $\mathcal{P}_\alpha$.
2. Determine the F-cost (see Sec. 4) and C-cost (see Sec. 5) of each arc in the net structures, so that: (a) the F values are accurately captured by corresponding flow costs, and (b) the option selection consistency requirements are satisfied.
3. Connect net structures to form the TSG that mirrors the topology of net connections in $\mathcal{P}_\alpha$ (e.g., see Fig. 3(b)).
4. Connect the TSG to the DPG via placement injection arcs so that for each cell in a net structure the amount of flow equal to its chosen transform option size is sent to its chosen global placement position in the DPG (see Sec. 7).
5. Translate the given total cell size constraint to a white-space constraint on the detailed placement process that is satisfied by flows through the DPG using the algorithm of [5].
6. The resulting arc cost formulation of Step 2 is a concave function. Determine the min-cost flow using the near-optimal network flow concave-cost minimization algorithm of [10].
7. Apply all the transforms chosen by the resulting flow to each cell and net in $\mathcal{P}_\alpha$ to obtain a near-optimal F for the circuit while satisfying the given total cell area constraint.

Figure 4: *Algorithm for post-placement selection of transform choices using min-cost network flow to obtain a near-optimal solution for a given objective function.*

1) An objective-function-dependent arc cost (F-cost) needs to be determined so that a flow cost accurately captures the delay objective function value (in general, any given objective function value) corresponding to the options chosen by the flow. Our solution for this issue is given in Sec. 4.

2) The flow that is determined must be a *valid flow*, i.e., can be converted to a valid set of transform option selections. Clearly, option nodes of a particular transform set have to be chosen in a mutually-exclusive manner. Furthermore, transform options of cells connected to multiple nets need to be included in each of the corresponding net structures in order to capture the objective function cost corresponding to each net. In such cases, the selected transform options for the common cell must be *consistent* across all these net structures. Net spanning structures and an auxiliary objective-function-independent arc cost for consistency (C-cost) are used to ensure a valid flow; these are discussed in Sec. 5.

3) The flows sent to the DPG should be of amounts equal to the cell sizes of the chosen transform options, and sent to the correct placement positions in the DPG that are determined by the chosen placement options. This issue is discussed in Sec. 7.

We note at this point that an arc e in our network flow graph with a total cost $cost(e)$ incurs a binary cost for a flow f through it, i.e., cost incurred is $cost(e)$ if $f > 0$, and 0 otherwise. This cost incurring formulation is different from that in a the "standard" network graph in which the incurred cost of a flow on an arc is a linear function of the flow amount. The binary cost formulation is needed in the discrete optimization scenario of the transform option selection problem, since an option is either selected ($f > 0$) or not ($f = 0$). This results in the incurred arc cost being a concave function of the flow f, and such a problem can be near-optimally solved using the algorithm in [10]. The high-level pseudo code of our method FlowSynth is given in Fig. 4.

4 Net Structures and F-cost Determination

We discuss here net structures for accurate F-cost determination.

4.1 A First Attempt: Linear Net Structure

Each net structure, as a mini TSG, has a *supply node* to send flows to, and a *gathering node* to gather the "remaining" flows from the transform option nodes and send them to child net structures; see

Fig. 5. Transform option nodes lie between the supply and gathering nodes. An intuitive way to connect the transform option nodes in a net structure is to sequentially connect the transform option sets. The resulting linear graph is shown in Fig. 6. For simplicity, we only show transform option nodes of four transforms in the figure: cell resizing $O_{res}(u)$, cell replacement $O_{pl}(u)$ and cell replication $O_{rep}(u)$ of a cell u in the net, and the type-1 buffer insertion O_{b1}; option node set O_{b2} for type-2 buffer insertion is not shown, but considered in our algorithm. For each transform, two options are shown (1 and 2).

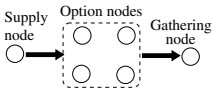

Figure 5: *High level view of a net structure.*

We now expose the difficulty of determining the F-cost of arcs in a linear structure in order to accurately captures the objective function (in this section, by cost of an arc we will only mean F-cost). We introduce an option selection variable s_x for each transform T_x. $s_x = k$ means that the k'th option O_x^k of transform T_x is chosen. Then, we can rewrite parameters in the delay objective function as functions of s_x's, e.g., $R_d = R_d(s_{res}(u_d), s_{b1})$ which means that the driving resistance R_d is determined by which driving cell-resizing and type-1 buffer insertion options are chosen. Similarly, we have $C_i(s_{res}(u_i), s_{rep}(u_i), s_{rep}(u_d))$[1], $l_{d,i}(s_{pl}(u_d), s_{pl}(u_i))$, etc., where recall that C_i is the input capacitance of cell u_i, and $l_{d,i}$ is the interconnect length between driving cell u_d and a load cell u_i. A flow through a net structure determines all s_x values. In order to achieve equivalence between the flow cost and the corresponding value of the delay objective function F_t (Eqn. 5), each product term in F_t needs to be part of the cost of one or more arcs on the flow path.

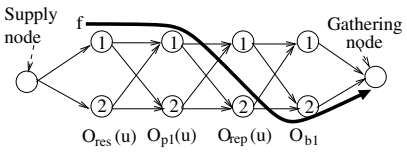

Figure 6: *A linear net structure.*

We define the *order* of a product term (henceforth only referred to as term) in F_t to be the number of option selection variables that the term is dependent on. An example of a high-order term is $R_d(s_{res}(u_d), s_{b1}) \cdot C_i(s_{res}(u_i), s_{rep}(u_i), s_{rep}(u_d))$, which is of order 5. For order-1 terms $P_x(s_x)$ which includes only one option selection variable s_x, the term $P_x(k)$ contributes to the cost of each arc emanating from option node O_x^k. For order-2 terms $P_{x,y}(s_x, s_y)$, if transforms T_x, T_y are adjacent in the linear structure, then the cost term $P_{x,y}(k,l)$ contributes to the cost of arc (O_x^k, O_y^l). On the other hand, for a term $P_{x,z}(s_x, s_z)$ for transforms T_x, T_z that are not adjacent in the linear net structure, like transforms T_{b1} and $T_{res}(u)$ shown in Fig. 6, it is impossible to accurately capture the term $P_{x,z}(k,m)$ in the cost of the flow arcs, since no arc connects (O_x^k, O_z^m). Furthermore, for a flow going through O_x^k, it is not known at the outgoing arc from O_x^k of the flow which option O_z^m will be chosen by the flow. Thus even if we assign a $P_{x,z}(k,\_)$ based cost to the outgoing arcs, it cannot be an exact one corresponding to a flow that chooses O_x^k and an arbitrary O_z^m.

For terms of order larger than two, it is impossible to assign costs to arcs connected to the option nodes of the transforms in the term so that the value of the term is accurately captured for any choice of options, even if some pairs of transforms are adjacent in the linear structure. For example, for an order-3 term $P_{x,y,z}(s_x, s_y, s_z)$ with $x = res(u)$, $y = pl(u)$ and $z = rep(u)$, and the linear net structure of Fig. 6, it is not known at an arc (O_x^k, O_y^l) which arc out of O_y^l to an O_z option will be chosen. Thus the value of $P_{x,y,z}(k,l,m)$ cannot be accurately captured by assigning values corresponding to $P_{x,y,z}$ to arcs connected to option nodes for these transforms.

4.2 Overlapping-Star-Graph Net Structure

As exemplified in the previous subsection, accurately capturing costs corresponding to complex non-linear objective functions is difficult to achieve with simple linear net structures. We thus need to have more complex net structures that mirror the terms in the objective function so that arc costs are accurate. To that end, we propose an *overlapping star graph structure*.

[1]Choosing a replication option for the driving cell u_d of a net n_j divides it into two subnets. If u_i is a critical sink and falls in the subset not driven by the "original" u_d, then in the net delay function (inner summation of Eqn. 5) for the subnet that u_d drives, C_i changes to zero. Therefore, $s_{rep}(u)$ can change the C_i component of the subnet delay function.

In an overlapping star graph structure, an order-t term ($t > 2$) in the objective function associated with t transforms $T_1, \ldots, T_t$ is converted to a *star graph* in the net structure with an arbitrary transform node set O_c as its center with arcs to the other $t-1$ transforms. One of the non-center transforms O_p is designated as the *parallel* transform; the other transforms are simply termed as *regular* transforms. In this star graph there are *m parallel arcs* between each option node pair between O_c and O_p, where $m = \prod_{O_i \neq O_c, O_p} k_i$ and k_i is the number of options in transform set O_i; thus this product is the number of combinations of the regular transform options. An example is shown in Fig. 7(a) for $t = 3$ and the 3 transform sets O_x, O_y, O_z, with $O_c = O_x$, $O_p = O_y$ and $k_z = 2$. The arcs between transforms in Fig. 7(a) are meta arcs, and are converted to complete bipartite subgraphs between the transform option sets in the net structure as shown in Fig. 7(b) for $k_x = k_y = k_z = 2$.

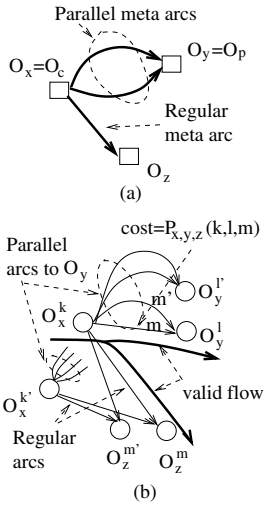

Figure 7: *A star graph structure for an order-3 term associated with transforms T_x, T_y and T_z. (a) The meta star graph. (b) Detailed star graph and a valid flow through it.*

The idea of parallel arcs is that each arc in a *parallel arc set* between 2 option nodes O_c^k, O_p^l represents exactly one option combination of the center, parallel and regular transforms in the star graph. Hence the cost of a *parallel arc* in the net structure can be assigned the value of objective function term $P_{O_1,\ldots,O_t}(i_1, \ldots, i_t)$ corresponding to option choice $O_j^{i_j}$ for each option set O_j. Thus for the above order-3 term example, there are 2 parallel arcs between the 2 option nodes $(O_c^k, O_p^l) = (O_x^k, O_y^l)$ of the center and parallel transforms. The parallel arc that represents the option choice O_z^m of the regular transform O_z has cost $P_{x,y,z}(k,l,m)$ corresponding to term $P_{x,y,z}$. The cost of non-parallel arcs between the center and regular transform options is 0.

The crux of the idea for making this star-graph structure work is to ensure that the flow through each star graph chooses a set of "consistent arcs" in it. For a star graph of an order-t term, a *consistent arc set* is defined as an arc set that contains one parallel arc and $t-2$ regular arcs from a center transform option node, and the regular transform option selections represented by the parallel arc is the same as the selections corresponding to the $t-2$ regular arcs.

Thus in Fig. 7(b), a consistent arc set is the parallel arc between O_x^k, O_y^l that represents the choice O_z^m of the regular option set O_z, and the regular arc (O_x^k, O_z^m). The cost of a flow through a consistent arc set S in the star graph = the cost of the parallel arc in it that it passes through = the term $P_{O_1,\ldots,O_t}(i_1, \ldots i_t)$ corresponding to transform options $O_1^{i_1}, \ldots, O_t^{i_t}$ that are the nodes in the consistent arc set (and thus selected by the flow). Our technique for ensuring that the flow through the star graph structure selects only a consistent arc set is discussed in Sec. 5. The capacity of each arc in a star graph will also be explained in that section. We note that in spite of the fact that each parallel arc a_p represents a combination of all transform options represented by the star graph, it is necessary for the flow to go through the entire consistent arc set containing a_p. One of the reasons for this is to inject flows into the DPG for each selected transform O_x^i in the star graph so that the cell corresponding to O_x^i can be placed, and its detailed placement cost considered, as part of the option selection process.

The various star graphs generated from each term of order larger than 2 are then merged by overlapping the same transforms in these terms to form the overlapping star graph structure. Figure 8 shows an example of an overlapping star graph structure for a net n_j with a driving cell u, and two critical sink cells v and w as shown in Fig. 8(a). For clarity of exposition, we do not show all the transforms considered in our actual implementation, but choose the following representative transforms: gate sizing and replication, and type 2 buffer insertion. In F_t (Eqn. 5), the parameters affected by these transforms are the driving

resistance $R_d(s_{res}(u))$, the net WL $L_j(s_{rep}(u), s_{b2})$, the input capacitances $C_v(s_{res}(v), s_{rep}(v), s_{res}(u))$ and $C_w(s_{res}(w), s_{rep}(w), s_{res}(u))$ of v and w, respectively. Accordingly, the terms with order greater than 2 in the objective function are $2c \cdot R_d L_j$ (order 3), $2R_d C_v$ (order 4), $2R_d C_w$ (order 4). Note that the coefficient of 2 in these terms indicates that this term is in the delay expression (Eqn. 4) of the interconnects from u to both v and w, and thus appear twice in F_t.

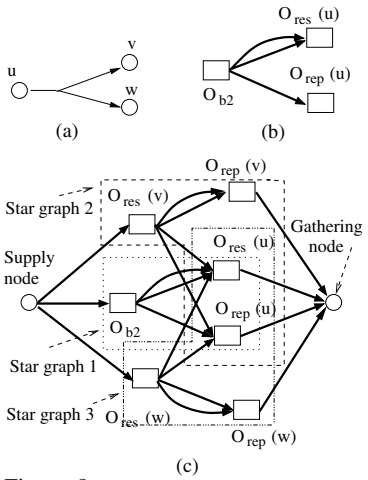

Figure 8: *(a) Net n_j. (b) Star graph 1 for term $2c \cdot R_d L_j$. (c) The overlapping-star-graph structure corresponding to the 3 terms in F_t that are dependent on the transforms considered in this example.*

Corresponding to the three terms, three star graphs are generated: star graph 1 corresponds to the term $2c \cdot R_d L_j$, star graph 2 corresponds to the term $2R_d C_v$, and star graph 3 corresponds to the term $2R_d C_w$. Star graph 1 is shown in Fig. 8(b), and the merged overlapping star graph structure is shown in Fig. 8(c).

For an order-2 term, no parallel arcs are needed. We can simply connect the two transform option sets O_x and O_y associated with the term by a bipartite subgraph similar to that in Fig. 6. Then, we add the term value $P_{O_x,O_y}(i,j)$ to the cost of each arc (O_x^i, O_y^j) between the two option sets. The order-1 terms are taken care of in the same way as described for the linear structure.

5 Consistent Option Selection

As we mentioned earlier, for a flow to translate to valid transformations of $\mathcal{P}_\alpha$, it must pass through: 1) only one option node for each transform set of a cell (mutual exclusiveness), and 2) the same option nodes in multiple net structures for each transform of a common cell in these net structures. Furthermore, for a flow to incur a correct cost, it must pass through a consistent set of arcs in each star graph structure. The mutual exclusiveness in option set selection is implied by the consistent arc set condition, and addressed in Sec. 5.2. Below we describe techniques for ensuring the second type of consistency.

5.1 Consistency Between Multiple Net Structures

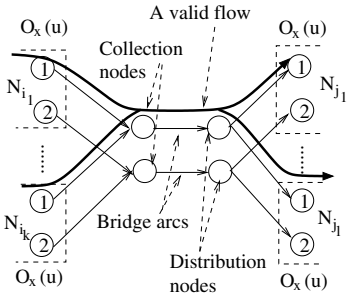

Figure 9: *The spanning net structure for maintaining consistency of option selection for a cell u that is common to multiple net structures.*

Let u be a common cell of multiple nets. Among these nets, u is a sink cell of k nets $n_{i_1}, \ldots, n_{i_k}$, and the driving cell of l nets $n_{j_1}, \ldots, n_{j_l}$. The corresponding net structures are shown in Fig. 9. These net structures are connected by a net spanning structure. Each of the above net structures will contain the corresponding option set $O_x(u)$; see Fig. 9. For consistency, we need to make sure that the selected options for each $O_x(u)$ are the same across all these net structures. Among the transform types we consider, the option choices for cell-related transforms, sizing, replication and replacement need to be consistent.

Let us first consider consistency among the parent net structures $N_{i_1}, \ldots, N_{i_k}$, where N_{i_m} is the net structure of net n_{i_m}. To maintain consistency, a set of bridge arcs are added in the spanning structure as shown in Fig. 9. Each *bridge arc* corresponds to one option node $O_x^i(u)$ of $O_x(u)$, and the $O_x^i(u)$'s in all these net structures $N_{i_1}, \ldots, N_{i_k}$ are connected to a *collection node* at the start end of its corresponding bridge arc. Hence, if in N_{i_m} the flow passes through $O_x^i(u)$, part of the flow will be diverted to the corresponding bridge arc (the other parts go to the DPG [see Sec. 7] and to the flow gathering node of N_{i_m}). Therefore, the problem of maintaining consistency among $N_{i_1}, \ldots, N_{i_k}$ is translated to ensuring the mutual exclusiveness in the bridge arc set, i.e., only one bridge arc can have flow on it.

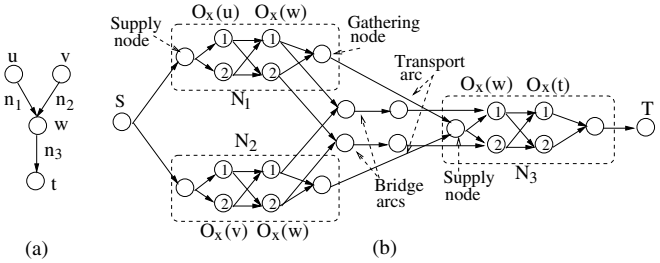

Figure 10: *(a) Interconnections between 3 nets. (b) The complete TSG corresponding to [a]. Transport arcs send the remaining needed flow to the supply nodes of child net structures that cannot be sent through the bridge arcs.*

For consistency between the parent net structures N_{i_m}'s and the child net structures N_{j_r}'s, we connect a *distribution node* at the termination end of each bridge arc to the same option of $O_x(u)$ in each N_{j_r} that the collection node of the bridge arc is connected to. In this way, if an option $O_x^i(u)$ is chosen (passed by flow) in N_{i_m}, the flow will also be sent to $O_x^i(u)$ in each N_{j_r} through the corresponding bridge arc. Thus, if mutual exclusiveness of the bridge arc set for O_x is ensured, the same options will be selected in each net structure to which u belongs as shown in Fig. 9. Techniques for satisfying the mutual exclusiveness condition for an arc set are discussed in Subsec. 5.2

The F-cost of all arcs in the spanning structure is 0. If $k > l$, the capacity of arcs from option nodes to collection nodes is 1, the capacity of arcs from distribution nodes to option nodes is k/l; otherwise for $k \leq l$, the capacities for the two sets of arcs are reversed. The capacity of each bridge arc is $\max\{k,l\}$. At this point, we give a simple example of a complete TSG in Fig. 10. For clarity, in each net structure only options of one transform T_x are shown.

5.2 Ensuring Mutual Exclusiveness and Consistent Arc Set Conditions

The standard min-cost network flow solves a linear programming problem (a continuous optimization problem), and thus cannot directly handle the discrete requirements in our method, such as the mutually exclusiveness of bridge arcs, and the selection of a consistent arc set in a star-graph structure. The basic idea we use to cope with this problem is to attach an objective-function-independent cost, C-cost, to those arcs for which any of the above discrete requirements has to be satisfied, i.e., arcs in star graph structures and bridge arcs. Then, the total cost $cost(e)$ for such arcs is $C(e) + C'(e)$, where $C(e)$ is the F-cost, and $C'(e)$ is the C-cost, while for other arcs $cost(e)$ is only $C(e)$.

Let C_Δ be the cost difference between the "continuous" min-cost flow (which is usually an invalid one for our problem) of cost C_{min}, and a valid flow of cost C_{val} in the network flow graph (TSG $\cup$ DPG) with only $C(e)$ as the arc cost. We determine $C'(e)$ according to two requirements: (1) the total C-cost C'_{val} incurred by any valid flow is the same, and (2) it is smaller than the total C-cost C'_{inval} incurred by any invalid flow by at least $C_\Delta + 1$. Then, in the network flow graph with both $C(e)$ and $C'(e)$ costs, the min-cost valid flow of cost $C^{min}_{val} + C'_{val}$ must also have smaller cost than any invalid flow, since for any invalid flow that incurs a C-cost of C_{inval}, its total cost is $C_{inval} + C'_{inval} \geq C_{min} + C'_{inval} \geq C_{min} + C'_{val} + C_\Delta + 1 \geq (C_{val} + C'_{val}) + 1 > C^{min}_{val} + C'_{val}$. Therefore, if $C'(e)$ can be determined to satisfy the above two requirements, then a min-cost flow in the network flow graph will be an optimal valid flow.

$C'(e)$ for ensuring the mutual exclusiveness condition To ensure the mutual exclusiveness condition in an arc set, $C'(e)$ is set as:

$$C'(e) = C_\Delta + 1 \tag{6}$$

for each arc e in the set. It is easy to see that for any valid flow that only goes through one arc in the set, the incurred C' cost is always $C_\Delta + 1$. Otherwise, if an invalid flow passes through $k > 1$ arcs in the arc set, the incurred C' cost becomes $k(C_\Delta + 1)$ which is larger than the C' of a valid flow by at least $C_\Delta + 1$. Hence the two conditions for $C'(e)$ are satisfied.

$C'(e)$ **for ensuring the consistent set condition** In order to determine $C'(e)$ in a star graph structure for the consistent set requirement, we utilize a basic property of the network flow: the total capacity of the set of outgoing arcs from a node that have flow in them has to be larger than or equal to the incoming flow amount to the node.

When we construct a star-graph structure, we use the following guideline. *The capacities of arcs in a star-graph structure are determined in such a way that for each option node of the center transform, only the total capacity of a consistent set of arcs from the node is equal to the incoming flow amount to the node. Then, $C'(e)$ of each arc e in a star graph is made proportional to the capacity of e.*

The guideline ensures that if the outgoing flow from a center transform option node chooses an inconsistent set of arcs in the net structure, then the total capacity of the set has to be larger than the incoming flow amount to the option node. Hence with $C'(e)$ proportional to the arc capacity, flow that passes through an inconsistent set of arcs will always incur a larger C' cost than a valid flow through a consistent set of arcs. Next, we describe in further detail how we determine the capacity of arc e in a star graph and capacity-dependent cost $C'(e)$.

Base capacity and capacity-dependent $C'(e)$ Instead of directly determining the actual capacity of outgoing arcs in a star graph from a center transform option node O_c^i, that is dependent on the incoming flow amount to O_c^i, we first determine the "base capacities" of these arcs that is independent of the incoming flow amount. The *base capacity* $cap_b(e)$ of an arc e is a fraction between $(0,1)$, and is determined such that only the total base capacity of a consistent set of arcs from O_c^i can be 1. The actual arc capacity $cap(e)$ of an outgoing arc e from O_c^i that satisfies the above guideline is then determined as the incoming flow amount $f_{in}(O_c^i)$ to O_c^i multiplied by the base capacity of e:

$$cap(e) = f_{in}(O_c^i) \times cap_b(e), \text{ for } e = (O_c^i, O_x^j)$$

where O_x is a non-center transform of the star graphs. We note that the incoming flow amount to an option node is fixed for any valid flow that chooses the option (as we will see in Sec. 6); thus $f_{in}(O_c^i)$ is a constant.

We now show how to determine the base capacities. In a star graph structure, besides the option node sets of the center transform O_c and parallel transform O_p, we denote all the r option nodes across all regular transforms by $O_{reg}^1, \ldots, O_{reg}^r$.

Our base capacity assignment method uses the following property of prime numbers: for a set of prime numbers $p_1 < p_2 < \ldots < p_r$, $\sum_{i=1}^r e_i/p_i$ cannot be an integer unless each e_i is a multiple of p_i [2]. Furthermore, we will choose this set of prime numbers in such a way that for any $j \le r$:

$$\sum_{i=1}^{j} 1/p_i < \frac{p_j - 1}{p_j} \tag{7}$$

The base capacity is assigned as follows. Determine prime numbers $p_1, \ldots, p_r$ so that Ineq. 7 is satisfied. Base capacities of arcs from nodes in O_c to O_{reg}^i $(1 \le i \le r)$ is set to be $(1/p_i)$, and the base capacity of an arc a_p in parallel arc set between O_c and O_p is set to be $(1 - b_1(a_p)/p_1 - \ldots - b_r(a_p)/p_r)$, where $b_i(a_p) = 1/0$ indicates if the arc corresponds to choosing option O_{reg}^i or not. For a subset S of arcs from a center transform option node to regular transform nodes, we define a vector $v(S) = (c_1, \ldots, c_r)$, where $c_i = 1/0$ if the arc to option O_{reg}^i is in/not in S. An example of base capacity assignment is shown in Fig. 11. The two arcs to the regular transform option are assigned base capacities of $1/3$ and $1/5$. Note that though 2 is also a prime number, it does not satisfy Ineq. 7. The upper of the two parallel arcs between O_c^i and O_p^j corresponds to choice O_{reg}^1, and thus has a base capacity of $1 - 1/5 = 4/5$. The bottom parallel arc corresponds to choice O_{reg}^2, and thus has a base capacity of $1 - 1/3 = 2/3$.

Theorem 1 *If we set the arc base capacity in a transform star graph according to the method stated above, then an inconsistent set of arcs will not have a total base capacity of 1, while any consistent set of arcs will have a total base capacity of 1.*

Proof: Let S be the subset of arcs from a center transform option node to regular transform nodes in which there are flows, and let S's vector be $v(S) = (c_1, \ldots, c_r)$. If the inconsistent arc set = S, then from the given property of prime numbers, its total base capacity $\sum_{i=1}^r c_i/p_i$ cannot be 1. Otherwise, for an inconsistent set $\{S \cup 1 \text{ parallel arc } a_p\}$, its total base capacity $1 - \sum_{i=1}^r (b_i(a_p) - c_i)/p_i$ cannot be an integer since not all $b_i(a_p) = c_i$ (otherwise it becomes a consistent set). For example, if we take the arc set of the upper parallel arc and the arc to O_{reg}^2 in Fig. 11, their total base capacity is $4/5 + 1/3 \ne 1$. Finally, if the inconsistent set contains $m > 1$ parallel arcs $a_{p,1}, \ldots, a_{p,m}$ besides S, then the total base capacity cap_b is:

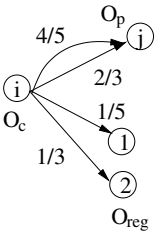

Figure 11: *An example of base capacity assignment.*

$$cap_b = m - \sum_{i=1}^{r} (d_i - c_i)/p_i \tag{8}$$

where $d_i = \sum_{k=1}^m b_i(a_{p,i})$; thus $m \ge d_i \ge d_i - c_i$. If any $d_i - c_i$ is not a multiple of p_i, then cap_b in Eqn. 8 cannot be an integer, and thus is not equal to 1. Otherwise, if each $d_i - c_i$ is a multiple of p_i, then at least one $d_i - c_i$ is not 0 in order that the $\sum_{i=1}^r (d_i - c_i)/p_i$ term in Eqn. 8 is not 0. Let i' be the maximum index for which $d_i - c_i$ is a non-zero multiple of p_i; then we have $(d_{i'} - c_{i'}) \ge p_{i'}$. Therefore, $\sum_{i=1}^r (d_i - c_i)/p_i = \sum_{i=1}^{i'} (d_i - c_i)/p_i \le \sum_{i=1}^{i'} m/p_{i'}$. According to Ineq. 7, we can further obtain $m \sum_{i=1}^{i'} 1/p_{i'} < m \cdot \frac{p_{i'}-1}{p_{i'}} \le m - 1$, since $m \ge d_{i'} - c_{i'} \ge p_{i'}$. Substituting this result into Eqn. 8, we get $cap_b > m - (m-1) = 1$. E.g., for an arc set of only the two parallel arcs in Fig. 11, its total base capacity is $4/5 + 2/3 > 1$. For a consistent set of arcs including a parallel arc a_p, the total base capacity is $1 - \sum_{i=1}^r (b_i(a_p) - c_i)/p_i$. Since $b_i(a_p) = c_i$ in a consistent set, its total base capacity is 1. $\diamondsuit$

After obtaining the base capacities of arcs in a star graph, we can determine the C' cost of these arcs. Let Δcap_b^{min} denote the minimum total base capacity difference between consistent and inconsistent sets of arcs[2]. $C'(e)$ of an arc e in a star graph structure is made proportional to the base capacity of e (and thus to the capacity of e), and is determined as:

$$C'(e) = \frac{C_\Delta + 1}{\Delta cap_b^{min}} cap_b(e) \tag{9}$$

With this $C'(e)$ formulation, the total C' cost incurred by any valid flow in the star graph through a consistent set of arcs is $\frac{C_\Delta+1}{\Delta cap_b^{min}}$ (since $\sum cap_b(e) = 1$ for arcs e in a consistent arc set), while the C' of an invalid flow through an inconsistent set of arcs is at least $\frac{C_\Delta+1}{\Delta cap_b^{min}} \cdot (1 + \Delta cap_b^{min})$, which is larger by an amount of $C_\Delta + 1$. Therefore, the two conditions for $C'(e)$ are satisfied. We thus have the following result.

Theorem 2 *The min-cost flow in the network flow graph with both costs $C(e)$ and $C'(e)$ as determined by Eqns. 6 and 9 is the same as the min-cost valid flow in the network flow graph with only cost $C(e)$, and thus provides the optimal valid transform selections in $\mathcal{P}_\alpha$ for the objective function.*

Proof: Follows from the discussion in this subsection. $\diamondsuit$

We note that obtaining a min-cost flow in a network with a concave arc cost (which, as discussed earlier in Sec. 3.2, is the type of network the TSG is) is NP-hard, and we solve the problem with an approximation algorithm proposed in [10] to obtain a near-optimal solution. Thus, while we do not find the optimal min-cost flow in our network graph as assumed in Theorem 2, the theorem also implies that a near min-cost flow results in a near-optimal transform option selections in $\mathcal{P}_\alpha$. This is also supported by our experimental results (Sec. 8), which show that our solution quality is high and significantly better than that of the current state-of-the-art.

6 Flow Amounts to Net Structures

To convert the base capacity of an arc to its actual capacity, we need to first determine the amount of flow that should be sent to each net structure in a valid flow. A net structure has two types of outgoing flows: 1) a *local flow* that goes into the DPG and sink for performing detailed placement of chosen synthesis options (see the next section

[2]We approximate this as the minimum of the following 2 quantities: (i) the smallest base capacity of an arc; and (ii) the smallest difference between the base capacities of any 2 arcs in the star graph.

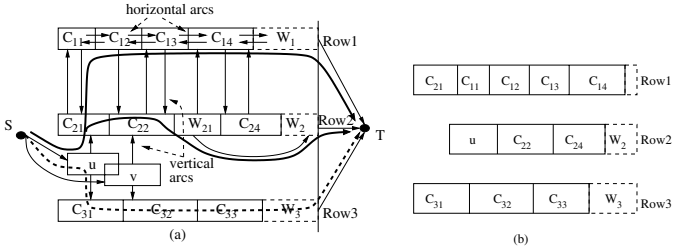

Figure 12: *(a) General n/w flow graph and flows for legalizing placement of cell u. C_{ij}'s are the fixed row cells, u and v are the moved cells placed in global placement, W_{ij}'s are the available white space (WS). Two alternative legalizing flows are shown in solid and dashed lines respectively. (b) The resulting position corresponding to the solid flows in (a), which legalize u. White spaces in Row 1 and Row 2 are decreased.*

for more details); 2) a *global flow* that is supplied to its child net structures for their flow amount requirements. For local outgoing flows, the amount of flow leaving at each option node set $O_x(u)$ on a cell u is a constant value $f_{x,max}$ that is proportional to the maximum possible size of u provided in the library (see the next section). Simply put, the incoming flow amount $f_{in}(N_i)$ of a net structure N_i has to be able to supply both types of outgoing flows, and thus satisfy:

$$f_{in}(N_i) = \sum_{O_x \in N_i} f_{x,max} + \sum_{N_j \in child(N_i)} f_{in}(N_j)/d_{in}(N_j) \qquad (10)$$

where $child(N_i)$ is the set of child net structures of N_i, and $d_{in}(N_j)$ is the incoming degree of N_j, i.e., the number of parent net structures of N_j. This recursive formulation is consistent with our design in which incoming flow amount for a net structure is sent uniformly from all its parent net structures. Based on the recursion in Eqn. 10, we determine the required incoming flow $f_{in}(N_i)$ to N_i recursively in reverse topological order, starting with leaf net structures that have no child net structures and are directly connected to the sink. The recursion ends at the source node S which is connected to the net structures corresponding to "beginning" or input nets of the paths in $\mathcal{P}_\alpha$.

7 Incorporating Detailed Placement with Transform Option Selection

A min-cost network flow based detailed placement algorithm for standard cell design was proposed in [5]. Fig. 12 shows the network graph structure for detailed placement (the DPG). The arcs in the graph model the possible movements of their start cells. For an illegal cell u, there are arcs connecting it to the two nearest legal position in the two adjacent rows. The flow in the graph depicts the amount of cell area moved in the arc direction. The illegal cells are connected to the source cells, while available white spaces are connected to the sink.

By pushing a flow from the source with amount equal to the illegal cell area into each illegal cells, then to their nearest legal positions, then to legally placed overlapping cells, and finally to the sink from white spaces, we can achieve the legalization of the placement with minimum cost (the unit flow cost of arcs is set to be the delay objective function change when the start cell of the arc is moved a unit distance in the arc direction) if all cells are moved in accordance to the flow.

In our synthesis transforms, we may change the position and size of the driving and load cells of a net, and add buffers and replicas; thus a detailed placement is also needed for legalizing these modified or added cells. The detailed placement can be performed at the same time as the transform selection process so that the delay cost of detailed placement will not be ignored when selecting transform options, and so that the total cell size increase constraint can be converted to an available WS amount constraint in the DPG. This is accomplished by connecting the DPG to the TSG by placement injection arcs.

Each placement injection arc $e^i_{inj}(u)$ from a net structure corresponds to a placement option $O^i_{pl}(u)$ of a cell u in the net structure. The added replica of a cell u is placed at the same position as u (the overlap is removed via flows in the DPG that perform incremental detailed placement). Similarly added buffers to a driving cell u_d are placed in the same position as u_d. $e^i_{inj}(u)$ starts from its corresponding placement option node in the TSG and ends at the position in the DPG determined by the option; see Fig. 13. Thus, the flow into the DPG is always sent to the correct position for a chosen placement option.

These incoming flows from the placement injection arc to the DPG replace the flows from the source in the detailed-placement-only scenario explained above. In order to perform correct detailed placement of the selected synthesis options via the DPG, the flow amount from $e^i_{inj}(u)$ to the DPG must be equal to the total size of all cells that are placed near u, i.e., u, its replica if any, and buffers placed near u, according to the chosen options in $O_{res}(u)$, $O_{rep}(u)$, O_{b1} and O_{b2} (we term these four transforms as *near-u transforms*). However, the options in the option sets of near-u transforms are chosen separately from the replacement option set $O_{pl}(u)$, and thus we need a structure to couple each near-u transform option set with $O_{pl}(u)$ so that the requisite flow amount, based on the chosen option sizes for near-u transforms, flows into the placement position chosen in $O_{pl}(u)$.

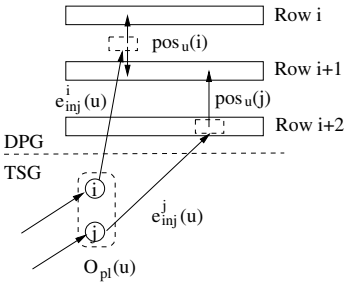

Figure 13: *Placement injection arcs from different placement options of u. $pos_u(i)$ denotes the global placement position of u determined by option $O^i_{pl}(u)$.*

The coupling is achieved by connecting $e^i_{inj}(u)$ with a *shunting structure* attached to each near-u transform option set, which is then connected to the sink. All arcs in the shunting structure have zero cost. An example of the shunting structure attached to option set $O_{res}(u)$ is shown in Fig. 14. The capacity of $e^i_{inj}(u)$ is equal to $\sum_{T_x(u) \in \text{near}-u \text{ transfom}} A_{x,max}$, where $A_{x,max}$ is the max cell size for transform $T_x(u)$. This allows enough flow to be sent to the DPG for any total size of these cells. However, if non-maximum sizes are chosen, not all flow on $e^i_{inj}(u)$ should be sent into the DPG. The function of shunting structures is to divert an appropriate amount of flow from $e^i_{inj}(u)$ directly to the sink so that the correct flow amount equal to the selected cell sizes is sent into the DPG. For simplicity, we discuss below the shunting structure for option set $O_{res}(u)$, and assume that the capacity of e^i_{inj} is $A_{max} = A_{res,max}$. The shunting structure for option sets of other near-u transforms are similar.

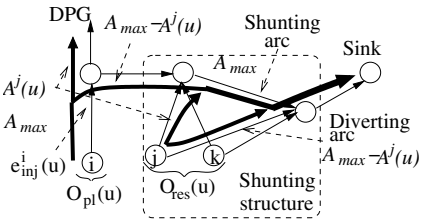

Figure 14: *The shunting structure attached to $O_{res}(u)$, and the amounts (in italics) of flows (shown dark) in the structure when $O^j_{res}(u)$ is chosen.*

The key component in the shunting structure is the shunting arc that connects the incoming arc to the shunting structure from e^i_{inj} to an outgoing arc from the shunting structure to the sink as shown in Fig. 14. Any flow that is diverted from e^i_{inj} to the sink has to pass through this arc. The capacity of the shunting arc is A_{max}. However, the available capacity of this arc for diverting flow from e^i_{inj} is modulated by the chosen option in $O_{res}(u)$. The modulation is achieved by connecting each option node $O^j_{res}(u)$ in $O_{res}(u)$ to the starting node of the shunting arc with an arc of capacity=$A^j(u)$, where $A^j(u)$ is the size of the chosen option $O^j_{res}(u)$. Thus, if $O^j_{res}(u)$ is chosen, a flow of amount $A^j(u)$ is sent to the shunting arc, which makes the remaining available capacity of the shunting arc $A_{max} - A^j(u)$. Since the shunting structure has 0-cost arcs, while the DPG arcs have positive costs, a flow of amount $A_{max} - A^j(u)$ is shunted from e^i_{inj}, which makes the amount of flow entering the DPG on the arc equal $A_{max} - (A_{max} - A^j(u)) = A^j(u)$, which is exactly the chosen size. Finally, a diverting arc is added from each option node $O^j_{res}(u)$ to the end node of the shunting arc with capacity equal to $A_{max} - A^j(u)$. This ensures that the total flow amount leaving the net structure at $O_{res}(u)$ is

Ckt	# cells	# cri. cells	Our method (FlowSynth) %ΔT	%ΔA	%ΔWL	runtime (secs)	Seq. appl. of transforms %ΔT	%ΔA	%ΔWL	runtime (secs)
td-ibm01	12K	266	28.0	-2.2	-11.3	633	20.1	-2.2	-9.8	275
td-ibm02	19K	584	29.7	-2.4	-12.6	628	24.6	-2.4	-10.5	338
td-ibm03	23K	425	44.2	-1.9	-8.8	993	37.1	-2.4	-8.9	529
td-ibm04	27K	559	27.8	-1.9	-12.4	1151	28.7	-2.1	-11.0	684
td-ibm05	28K	593	31.9	-2.5	-8.0	1134	23.0	-2.0	-8.5	649
td-ibm06	32K	572	37.7	-2.2	-11.2	1649	24.5	-2.0	-8.6	1124
td-ibm07	46K	705	33.4	-2.6	-8.6	1697	17.0	-1.9	-7.5	1134
td-ibm08	51K	694	39.0	-1.9	-7.2	1827	31.5	-1.7	-8.0	1055
td-ibm09	53K	889	34.9	-1.8	-10.2	2985	23.6	-2.0	-11.0	1722
td-ibm10	69K	912	23.2	-2.4	-8.1	3028	17.2	-1.9	-8.0	1803
td-ibm11	70K	910	31.5	-1.6	-10.5	3417	32.4	-2.7	-9.8	2154
td-ibm12	70K	916	38.4	-1.6	-10.8	2158	29.9	-1.5	-9.3	1536
td-ibm13	84K	889	36.2	-1.7	-9.4	3338	26.0	2.6	-8.6	1689
td-ibm14	147K	1185	22.4	-1.5	-7.5	4028	15.8	-1.6	-7.0	2199
td-ibm15	161K	1201	37.5	-1.4	-5.9	4087	25.8	-1.3	-5.7	3318
td-ibm16	183K	1234	38.2	-1.3	-7.0	5847	21.2	-1.1	-7.9	3542
td-ibm17	185K	1687	48.0	-1.3	-3.3	5812	32.3	-1.3	-5.5	3914
td-ibm18	210K	1779	46.0	-0.9	-3.3	5993	36.6	-0.9	-4.8	4378
Avg.		**888**	**34.8**	**-1.8**	**-8.7**	**2796**	**25.9**	**-1.7**	**-8.4**	**1780**
matrix	3.0K	201	9.5	-9.5	-10.2	1124	6.1	-9.9	-13.0	992
vp2	8.9K	218	13.7	-9.0	-10.2	1347	6.9	-9.9	-12.9	1038
mac32	25K	257	18.6	-9.2	-9.8	1397	11.1	-9.8	-11.9	1134
mac64	8.7K	314	18.4	-8.9	-7.6	1319	11.3	-10.0	-12.4	1208
Avg.		**248**	**15.1**	**-9.2**	**-9.5**	**1296**	**8.8**	**-9.9**	**-12.6**	**1093**
C432	160	50	16.8	-9.7	-11.4	334	8.5	-10.0	-14.5	129
C499	202	51	17.8	-10.0	-12.5	276	9.1	-10.0	-12.7	211
C880	383	77	19.5	-10.0	-11.2	248	9.5	-10.0	-13.1	231
C1355	544	85	15.9	-9.5	-9.6	382	10.6	-10.0	-12.0	244
C1908	880	88	24.1	-9.8	-12.1	376	14.9	-9.5	-12.3	223
C2670	1.3K	91	15.1	-9.2	-8.4	357	10.5	-9.8	-10.1	241
C3540	1.7K	124	25.8	-9.2	-11.6	501	15.6	-10.0	-13.3	338
C5315	2.3K	138	25.8	-9.3	-8.9	772	17.2	-9.5	-8.9	361
C6288	2.4K	299	25.4	-9.0	-7.5	859	16.9	-9.7	-10.0	593
C7552	3.5K	199	18.1	-9.3	-8.8	724	12.2	-9.9	-8.5	674
Avg.		**120**	**20.4**	**-9.5**	**-10.2**	**483**	**12.5**	**-9.8**	**-11.5**	**314**
Overall Avg.		**568**	**27.8**	**-5.1**	**-9.2**	**1885**	**19.6**	**-5.3**	**-9.9**	**1236**

Table 1: *Results for our method FlowSynth and for a sequential application of transforms. %ΔT is the percentage timing improvement, %ΔA and %ΔWL are the percentage changes of total cell area and WL, respectively (a negative value indicates deterioration). Cell area increase constraints on TD-IBM circuits is 3%, and is 10% for the other two benchmark suites since they have relatively smaller circuits.*

always a constant value of A_{max} irrespective of which option is chosen in $O_{res}(u)$.

8 Experimental Results

We used three benchmark suites in our experiments: 1) The TD-Dragon suite of [16], 2) ISCAS'85 benchmarks and 3) TD-IBM benchmark suite from [5]. For the first two sets of benchmarks, we use a $0.18um$ standard cell library, which provides four cell implementations with different areas, driving resistances, input capacitances and intrinsic delays for each function. For the IBM benchmarks, we do not have a corresponding cell library. Thus, the intrinsic cell delay is ignored for this benchmarks suite, and we set the driving resistance $R_d = 1440\Omega$ and input capacitance $C_g = 10^{-15} fF$ of each cell; these values are consistent with $0.18\ \mu m$ technology[3]. We consider five different implementations of a cell for cell resizing : $w/4$, $w/2$, w, $2w$ and $4w$ where w is the original width of the cell. Results were obtained on Pentium IV machines with 1GB of main memory.

The size of the benchmarks are shown in Table 1. We obtain results for our method FlowSynth that considers synthesis transforms simultaneously, and for a method that applies transforms in a good sequential order. In the latter case, we apply one transform at a time to all cells and nets in $\mathcal{P}_\alpha$ in the order of decreasing ratio of timing-improvement to area-increase of each transform. This provides the maximal benefit-to-cost ratio for a sequential approach, and, as indicated by our experiments, also provides the best delay improvements compared to other orders like best delay improvement. Thus we apply transforms in the order: (1) incremental placement, (2) type 2 buffer insertion, (3) cell resizing, (4) type 1 buffer insertion, and (5) cell replication. After applying all the transforms, we perform incremental detailed placement using the DPG to get a legal solution. The algorithm for incremental placement is from [5], which, as mentioned earlier, is also used in FlowSynth to perform incremental placement simultaneously with the synthesis option selection process in the TSG. The algorithm for buffer insertion is from [8], and the replication algorithm is from [13]. For cell resizing, we use a network flow technique with only cell-sizing options [12]. We compare this technique to an exhaustive enumeration sizing algorithm that gives the optimal solution. For small circuits on which we could perform exhaustive enumeration, the results produced by our resizing technique is only an absolute of 1% off the optimal value at $\frac{1}{60}$'th of its runtime [12], which establishes both the quality and efficiency of our network flow based cell sizing algorithm.

The results for FlowSynth and sequential transform applications are shown in Table 1. For TD-IBM benchmarks, we allow an extra 3% layout area compared to the original layout so that the total cell area increase is within 3%. For TD-dragon and ISCAS'85 benchmarks, this parameter is set to be 10% since these circuits are relatively small. We obtain a timing improvement of up to 48% and an average of 27.8% while satisfying the given area constraints. Our method is consistently better than the sequential application of transforms. The average timing margin between our method and the sequential application one is an absolute of 8.2%, and 40% relatively.

Furthermore, if we want to perform in-place physical synthesis, i.e., we do not want to change the position of cells in order to preserve interconnect-related metrics of the initial placement such as routability, WL and net switching power, then the placement option will not be considered. The average timing improvements in this scenario for FlowSynth and the sequential method are 18.9% and 12.6%, respectively, which implies a 50% relatively better performance by our technique. These results show that our algorithm is much better at finding global near-optimal choices than the sequential method. The WL and cell area increases are similar for both approaches. Our run time is only about 50% larger than that of the sequential approach.

9 Conclusions

We presented a novel network flow based method for applying multiple synthesis transforms in the post-placement stage for timing improvement by simultaneously considering these applications across transforms and across cells/nets of the "problem" parts of the circuit. We proposed a transform option selection graph (TSG) structure, in which transform options are modeled as nodes, and the cost of flows passing through various nodes is equal to the delay objective function value when the transforms corresponding to these nodes are applied. The TSG structure, based on overlapping star graphs, was obtained from the terms in the objective function that are dependent on the transforms we consider—this generalizes our method to any objective function and any transform set. We developed various novel and effective discretizing techniques to prevent invalid flows in the TSG (that would normally be obtained by a continuous optimization method like network flow), and thereby obtain a legal near-min-cost solution to our problem. We also coupled the TSG with a detailed placement graph, so that detailed placement is performed simultaneously with transform selection. The results show that timing improvements obtained by our method are significantly better than the improvements obtained by sequentially applying the set of transforms using the best benefit-to-cost order (which yields the best delay improvement), an approach that is comparable to those used by most state-of-the-art methods.

[3] Other electrical parameters we use are: unit length interconnect resistance and capacitance are $r = 7.6 \times 10^{-2}\,\mathrm{ohm}/\mu\mathrm{m}$, $c = 118 \times 10^{-18} f/\mu\mathrm{m}$.

References

[1] R. Ahuja, et al., "A Network Simplex Algorithm with O(n) Consecutive Degenerate Pivots", *Operations Research Letters (ORL)*, pp. 1417-1436, 1995.

[2] T. M. Apostol, *Introduction to Analytic Number Theory*, Springer, 1998.

[3] O. Coudert,"Gate sizing for constrained delay/power/area optimization", *IEEE Trans. VLSI, vol. 5, no. 4*, pp. 465-472, 1997.

[4] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy and A. Sullivan, "Transformational Placement and Synthesis", *DATE'2000*, pp. 194-201, 2000.

[5] S. Dutt, H. Ren, F. Yuan and V. Suthar, "A Network-Flow Approach to Timing-Driven Incremental Placement for ASICs", *ICCAD'06*, pp. 375-382, 2006.

[6] J. Fishburn and A. Dunlop, "Tilos: A posynomial programming approach to transistor sizing," *ICCAD'85*, pp. 326-328, 1985.

[7] S. Hu, M. Ketkary and J. Hu, "Gate Sizing For Cell Library-Based Designs," *DAC'07*, pp. 847-852, 2007.

[8] Y. Jiang, et al., "Interleaving Buffer Insertion and Transistor Sizing into a Single Optimization", *IEEE Trans. VLSI, Vol. 6, No. 4*, pp. 625-633, 1998.

[9] L. Kannan, P. Suaris and H. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization", *DAC'94*, pp. 327 - 332, 1994.

[10] D. Kim, and P. Pardalos, "A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure", *ORL*, pp. 195-203, 1999.

[11] A. Lu, H. Eisenmann and G. Stenz, "Combining Technology Mapping with Post-Placement Resynthesis for Performance Optimization", *ICCD'98* pp. 616, 1998.

[12] H. Ren and S. Dutt, "A Network-Flow Based Cell Sizing Algorithm", *The International Workshop on Logic Synthesis*, pp. 7-14, 2008.

[13] A. Srivastava, R. Kastner, C. Chen, and M. Sarrafzadeh. "Timing Driven Gate Duplication", *IEEE Trans. VLSI, Vol. 12, No. 1*, Jan. 2004.

[14] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree network for minimal Elmore delay," *Proc. IEEE Int. Symp. Circuits Syst.*, pp. 865-868, 1990.

[15] C. Wonjoon, K. Bazargan, "Incremental placement for timing optimization", *ICCAD'03*, pp. 463-466, 2003.

[16] Y. Xiaojian, C. Bo-Kyung and M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint generation", *ICCAD*, pp. 177-180, 2002.