Optimal Die Placement for Interposer-Based 3D ICs

Sergii Osmolovskyi[1], Johann Knechtel[2], Igor L. Markov[3], Jens Lienig[1]
[1] Institute of Electromechanical and Electronic Design, Dresden University of Technology, Germany
[2] Department of EECS, New York University Abu Dhabi, UAE
[3] Department of EECS, University of Michigan, USA
sergii.osmolovskyi@tu-dresden.de, johann@nyu.edu, imarkov@eecs.umich.edu, jens@ieee.org

Abstract— Performance of modern multi-chip modules, increasingly implemented as interposer solutions, is limited by system-level interconnects. We propose an effective method for optimal wirelength-driven die placement of interposer-based 3D ICs. Our key ideas are to leverage the constraint-satisfaction problem (CSP) formalism in combination with a branch-and-bound (B&B) algorithm, and to develop several novel techniques for early identification and pruning of unpromising configurations. Such techniques are crucial for addressing the combinatorial explosion when solving the NP-hard placement problem. Experiments on ISPD08 (modified) and MCNC benchmarks demonstrate that our method outperforms prior art: we can optimally place up to eleven rotatable dies, whereas state-of-the-art tools are limited to six dies.

I. Introduction

As transistor scaling reaches the limits of miniaturization and manufacturability [1], three-dimensional integrated circuits (3D ICs) offer a promising alternative [2–7]. 3D ICs comprise multiple dies (or active layers) which are either vertically stacked, monolithically integrated, or laterally placed on an *interposer*, i.e., a substrate with metal layers serving as a system-level integration platform (Fig. 1). Such interposer-based 3D ICs—also known as *2.5D ICs*—promise higher yield, simplified heterogeneous and system-level integration, as well as better heat dissipation compared to stacked or monolithic 3D ICs [3, 4, 6, 8, 9]. Further, they are commercially practicable, as demonstrated by various interposer-based 3D ICs on the market [10–12]. Designing such novel 3D ICs, however, typically requires some manual intervention for placement, simulation, verification, etc. [4,8,13]. That is, there is a need for dedicated design automation tools in order to meet the prospects of 3D integration.

When integrating multiple dies on an interposer, their placement should be optimized to reduce integration overhead and shorten system-level interconnects [4]. As interposers commonly integrate only a few dies (2–10), their optimal arrangement is within reach. Still, prior art is typically content with suboptimal solutions, e.g., Ho and Chang [14] as well as Seemuth *et al.* [15] leverage *simulated annealing*. The work of Liu *et al.* [16] is based on *enumerative search* and can, thus, place dies optimally. Despite the fact that the authors develop various heuristics for efficient computation, their approach does not scale beyond six dies in practice. As shown in our investigation (Section IV), we obtain comparable placement solutions and do so within $10^4\times$ shorter runtime in some cases. Closely related to optimal die placement, *optimal floorplanning* has been addressed by Onodera *et al.* [17] and by Funke *et al.* [18]. However, these studies were limited to six modules [17] or the rotation of modules was neglected [18], thus not guaranteeing optimal solutions. Overall, for relatively large die counts ($\geqslant$7), optimal solutions are hard to find but may offer significantly better placement quality than solutions obtained by heuristics.

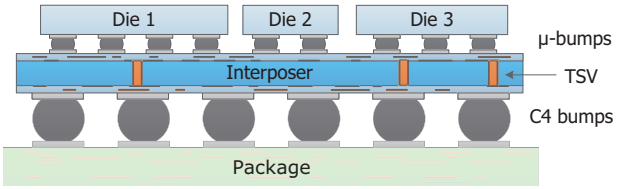


Fig. 1. Structure of interposer-based 3D ICs, also known as 2.5D ICs.

This work bridges the gap between optimal and heuristic placement in terms of performance and solution quality. We leverage the *constraint-satisfaction problem (CSP)* formalism from [19] and the parallel branch-and-bound (B&B) method. Upon this framework, we develop novel and highly efficient pruning techniques, where the coverage of the solution space is carefully relaxed to dramatically improve runtime without any sensible loss of solution quality. In fact, our pruning techniques reduce the search time significantly over prior art (Fig. 2) while still producing optimal placements for all available benchmarks. We empirically outperform all optimal placement methods in runtime and all heuristic methods in solution quality. Our technique can optimally place up to eleven dies while state-of-the-art tools top off at six dies.

In short, we make the following contributions:

1. We formulate optimal die placement as a constraint-satisfaction problem (CSP) and develop a parallel B&B algorithm for an efficient exploration of the combinatorial solution space.

2. We introduce effective pruning techniques that dramatically, yet accurately, confine the solution space and quickly rule out unpromising partial configurations. Thus, (i) we obtain optimal placement solutions in significantly shorter runtime than prior art, and (ii) we can optimally handle larger problem instances than prior art.

3. We make our tool publicly available [20], thus we enable the community to design optimally placed interposer-based 3D ICs.

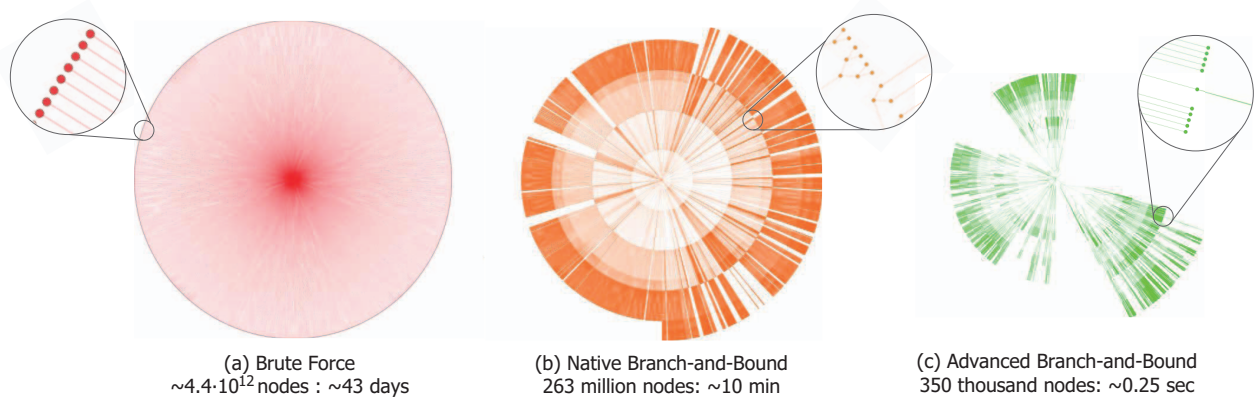


Fig. 2. Configuration spaces traversed by optimization algorithms on a benchmark with six dies. Each subfigure represents a search tree with branches extending from the root node in the center. Note the reduced search space of our approach due to novel pruning techniques (right).

II. Our Placement Framework

The main goal of our work is to overcome the limitation of prior art regarding the number of dies which can be placed optimally *and* efficiently. Towards this end, we propose a placement framework which involves two components: a branch-and-bound (B&B) method with novel pruning techniques, to address the combinatorial explosion of the problem for larger die counts, and a constraint-satisfaction problem (CSP) representation, to efficiently and accurately encode the layout.

We pursue die placement on the interposer with prefabricated chips, predefined interposer geometry and preset external-pin positions. Our **objective** is to find optimized positions and orientations for all dies while targeting for minimal total wire-length (TWL), which is calculated as the half-perimeter wire-length (HPWL). Additionally, valid placements must satisfy non-overlapping and containment constraints.

A. Layout Representation

We model die placement as CSP according to [19], but we also capture rotations of dies. While conceptually similar to horizontal and vertical constraint graphs, the formalism of [19] operates at two levels of abstraction: (i) conventional CSPs that represent the placement of a particular die configuration, and (ii) meta-CSP variables that modify such CSPs. This formalism helps us define a convenient search tree, supports automated reasoning about die configurations, and allows us to efficiently apply branch-and-bound techniques.

To represent a die placement, we distinguish three properties:

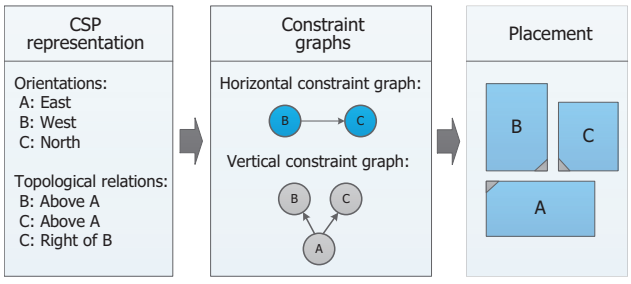


Fig. 3. An exemplary CSP encoding with the corresponding placement (CSP–constraint-satisfaction problem).

Orientation for each die can select one of four values: north ($0°$), west ($90°$), south ($180°$) and east ($270°$).
Containment constraints ensure that the dies do not extend beyond the placement zone on the interposer.
Topological relations represent the relative placement of two non-overlapping dies d_1 and d_2. There are four possible cases: d_1 left of d_2, d_1 right of d_2, d_1 above d_2, and d_1 below d_2.

For a placement instance with N dies encoded as CSP, non-overlapping die locations can be found in $O(N^2)$ time and space by constructing the horizontal and vertical constraint graphs, and tracing their directed paths (Fig. 3).

B. Our Branch-and-Bound Approach

Fundamentally, the B&B approach is a combination of two mechanisms: traversal of the search tree (branching), and estimation along with pruning of partial configurations (bounding).

During **branching**, we incrementally construct the placement according to a given schedule (see Sec. III-A). The following branching rules are applied (Fig. 4):

- Dies are added one at a time to the search tree.
- For each new die, we first create four nodes with all their orientations (blue nodes in Fig. 4).
- Then, we construct four topological nodes for each pair (new die, previously assigned die) of dies. These nodes are shown in gray in Fig. 4.

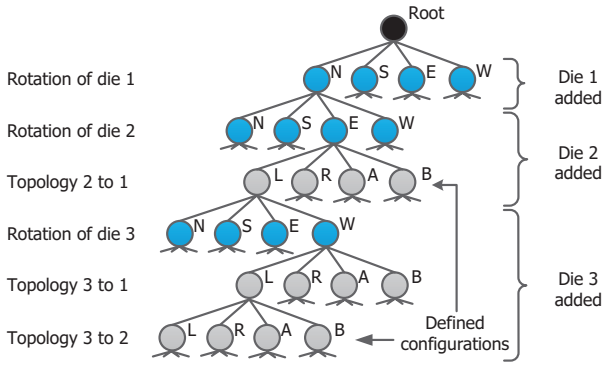


Fig. 4. Branching for the first three dies. Blue nodes indicate die rotations, gray nodes their topological relations. Defined configurations can be directly transformed into partial placements.

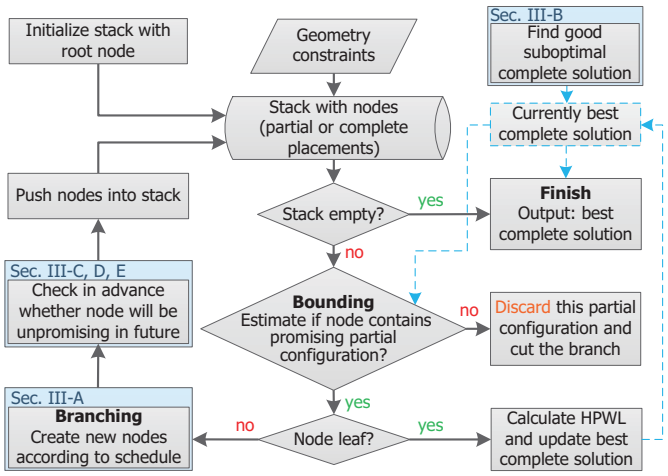


Fig. 5. Flowchart for the proposed placement framework.

We traverse the search tree using depth-first search. Before inserting any new nodes, we sort them according to their TWL estimation. Thus, we always consider the most promising nodes first (similarly best-first search).

Along with complete placement solutions (represented by leaves), the search tree contains many partial configurations that correspond to incomplete placements (nodes in Fig. 4). The latter may either be (i) directly converted into actual partial placements, followed by calculating the HPWL (for configurations where topological relations between all dies are *defined*), or (ii) heuristically estimated in terms of the TWL value (for configurations where some relations are yet *undefined*).

Bounding evaluates whether a partial configuration is promising or not. Towards this end, we estimate the lower bound of the TWL for the partial configuration (see Sec. III-C, D, E) against the TWL of the currently best known complete solution. In case the former is larger than the latter, the partial configuration cannot provide a better solution than the best one found so far and can be ignored without loss of placement quality. Thus, such partial configuration is rated as unpromising, the corresponding branch is pruned, and backtracking occurs.

The interaction between branching and bounding is outlined in Fig. 5. Initially we determine a high-quality, but suboptimal placement solution (upper-right corner in Fig. 5). This initial solution is used as reference for our branch-and-bound flow; it is essential to speed up the bounding/pruning process without loss of quality (Sec. III-B). We then initiate a container (stack) to hold all the nodes to be investigated (recall that each node represents a partial or complete placement configuration). For each node we first perform a bounding operation, i.e., we estimate whether it contains a promising configuration and discard it otherwise. Next, in case the promising node is a leaf, we accordingly update the best solution found so far. Otherwise, for regular nodes, we branch the tree and create four new nodes (recall Fig. 4). Before pushing new nodes into the stack we check whether they will become unpromising in the future using our pruning techniques (Sec. III). The above process is repeated until all nodes are processed. Once the stack is empty, all promising nodes have been investigated, and the currently best complete solution represents an optimal placement result.

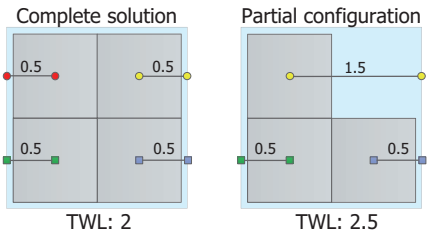


Fig. 6. Example of suboptimal placement of partial configurations due to lower-left-centric packing within whitespace.

C. Implementation of Parallel Branch-and-Bound

We implement our B&B algorithm using the *Shared Memory* model. Our implementation includes separated local containers (stacks) for each thread, and a shared common pool for nodes, thus leveraging the optimization potential of cache memories.

Initially, several nodes ($>$ number of threads) are generated by one of the threads within the shared memory. Then, each thread takes off one node and independently runs the B&B algorithm using its local container. If a thread has handled the full solution subset of its current node and the shared pool is also empty, it passes an "idle" message to all other threads. Any thread which catches this message swaps its local container with the shared pool in $O(1)$ time. Then, we repeat the process until all nodes in the whole search tree are checked.

D. Placement Optimization Within Whitespace

Since our CSP encoding (leveraging constraint graphs) packs dies towards the lower-left corner of the interposer by default, we may observe unoptimized placements when whitespace is available on the interposer. Moreover, as partial configurations will always contain some whitespace, along with potentially large displacements from optimal die locations, these configurations may exhibit an overestimated TWL. In turn, this may lead to pruning the related branches which may have eventually provided an optimal solution.

Consider Fig. 6 for an example. Here, each die is connected to a nearby terminal, inducing a TWL of 2 for the complete solution (Fig. 6[left]). Now, consider a partial configuration with one die less (Fig. 6[right]), as it occurs earlier during the B&B search. Due to the lower-left-centric packing, the top-right die will be moved to the left, thus increasing the TWL to 2.5. This overestimation of the TWL is potentially suggesting the B&B search to prune this part of the search tree, whereas the subsequently determined complete solution may be optimal nevertheless. In other words, such TWL overestimation may undermine the chances of finding optimal solutions.

To address this problem, accurate and fast techniques for placement optimization within whitespace are required. Prior art proposes a min-cost flow problem formulation [21], which slowed down our algorithms by a factor of 4 when being integrated. Therefore, we propose a fast analytical technique that moves dies one at a time within the whitespace, to iteratively find optimal locations. The idea here is to temporarily fix all dies except the one to be moved, define this die's placement zone with respect to the constraints in the CSP, and then derive an *optimal movement*. For the latter, we evaluate all the pins of

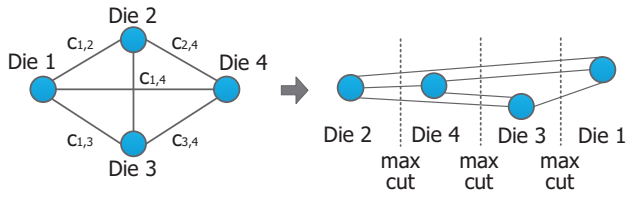


Fig. 7. Die ordering according to their impact on TWL.

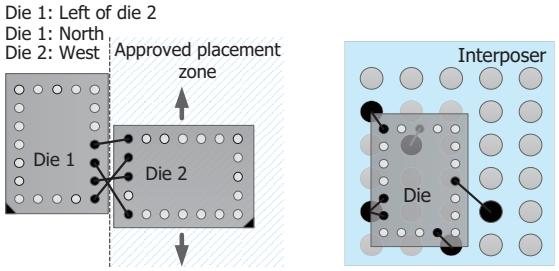


Fig. 8. Estimation of minimal WL (a) between a pair of dies for Forward Checking and (b) between each die and external interposer pins for Handling of Terminals.

that die in terms of which can be moved and how far, which shall remain fixed, and which can be moved within some confined region, all while targeting for minimal HPWL.

To optimally arrange the placement, we (i) move all dies together as one "virtual die," minimizing the terminal connections, and (ii) perform the technique outlined above for each die, in multiple iterations, until no further improvement is achieved.

III. Advanced Branch-and-Bound

The maximum number of dies that can be handled by native B&B algorithms (i.e., without complementary pruning techniques), is limited to six in practice. To overcome the combinatorial explosion for larger die counts, we propose, implement, and assess several novel and powerful placement-specific pruning techniques. These techniques allow for early identification of many unpromising configurations in the search tree, which would otherwise have to be evaluated in a native approach. In the following, we describe these techniques in detail.

A. Ordering the Branching Schedule

The order in which dies are added to the search procedure—also referred to as *branching schedule*—significantly affect the B&B search time. The sooner a die which greatly impacts the TWL is added, the more accurately the partial configurations can be estimated. In turn, we can detect and prune unpromising configurations early on. For example, assume that three out of ten dies contribute to 80% of the TWL; they dominate the overall placement and should be added first to the B&B search tree. Any unpromising arrangement of those three dies (inducing excessive TWL) can then be detected earlier, and the related branches of the search tree can be pruned.

We propose a greedy, graph-based algorithm to sort dies according to their impact on the TWL (Fig. 7). Unlike [17, 18], where dies are only sorted according to their size, we consider both the size as well as the interconnects of any die. For this, we construct a complete graph where each die is represented by a vertex and each edge between two dies (nodes) i, j holds a weight $c_{i,j} = (w_i + h_i + w_j + h_j)/2 \cdot n_{comm}$, where w_i and h_i are the respective width and height of die i, and n_{comm} is the number of nets connecting dies i, j. Then, we iteratively determine a node with the maximum cut to already sorted nodes.

This heuristic search scheduling cannot undermine optimality; it only impacts the computational efforts of the search, not the actual search result.

B. Initial Complete Solution

The sooner we obtain a high-quality, *complete* placement solution, the earlier we can prune unpromising partial configurations. For example, assume a partial configuration's TWL is estimated as 250mm while the currently best complete solution exhibits a TWL of 300mm. This partial configuration would be naturally designated as promising and further considered. However, assuming we can find a close-to-optimal complete solution early on, e.g., with a TWL of 200mm, then this partial configuration becomes unpromising and can be pruned.

As motivated, we seek to find high-quality compete solutions early on. To do so, we perform our optimization in two B&B runs. In the first run, we calibrate our B&B algorithm to overestimate the lower bounds of partial configurations. Thereby, many of them are purposely discarded, leaving only a few "dominant" and promising configurations. This allows for speedy determination of a good (but suboptimal) complete solution. We empirically observe that such solutions are close to the final optimal solution, with TWL deviations typically less than 10%. In the second run, we use that initial placement as the initially best solution for our regular B&B algorithm. We apply unscaled and accurate estimations for the lower bounds during that second run, thereby granting an optimal placement as a final result.

C. Forward Wirelength Checking

We can check in advance whether any topological constraint for a pair of dies will render the respective partial configuration unpromising *in the future*, even before we allocate and evaluate it. The idea is to predict the *minimal augmentation* of the TWL once we know the relative arrangement of two dies (e.g., die 1 is left of die 2). The minimal augmentation is captured by the HPWL for the nets connecting exclusively the two dies with each other, while considering the dies' optimal "back-to-back" placement (Fig. 8(a)).

To estimate how any topological relation will enlarge the TWL, we precalculate the minimal WL across every possible rotation and topology for each pair of dies. Accordingly, we vary the topology between the dies (left, right, above, below) and the rotation of both dies (north, west, south, east), resulting in 4^3 different variations for each pair. We place every variation as shown in Fig. 8(a) and minimize the HPWL using our analytical approach introduced in Sec. II-D. To ensure the minimal WL

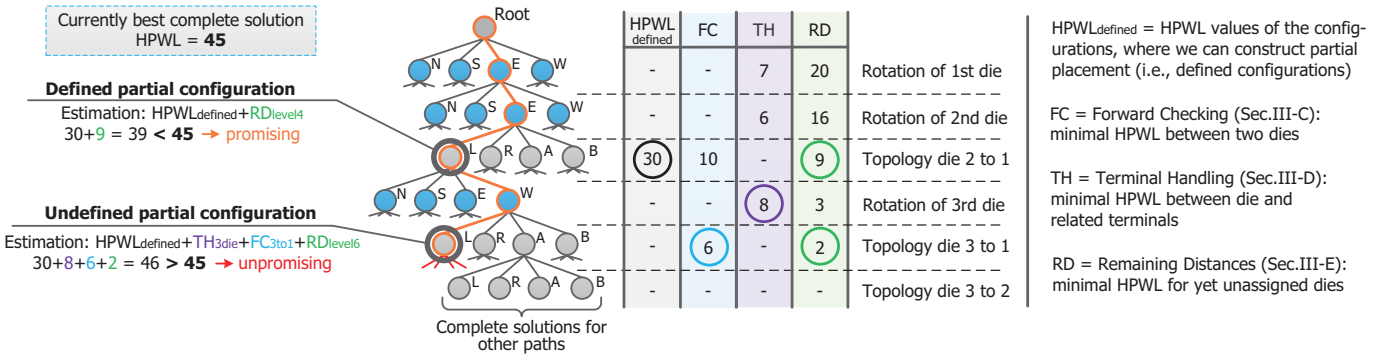


Fig. 9. Bounding of partial configurations using our proposed pruning techniques. The HPWL values are for the exemplary orange path in the search tree; values for other paths will be different. Besides pruning of unpromising nodes (e.g., the one related to the "Undefined partial configuration" in this example), the remaining nodes are investigated and branched as long as they are promising, until complete solutions can be derived, as indicated by the "Complete solutions for other paths."

between two dies, we consider only their common nets; terminal nets are to be considered separately (see Sec. III-D).

During branching, whenever we seek to assign a new topological node (gray in Fig. 4), we first check whether this is promising. We do so by comparing the currently best known solution against the sum of the current configuration's TWL estimation and the minimal WL precalculated for this new topology (as discussed above). In case it is promising, we assign the node and set its current TWL estimation to the above sum. When the configuration becomes defined after assignment of this node, we can further update the TWL estimation with the actual HPWL value. Unpromising branches are pruned.

D. Handling of Fixed Terminals

During forward checking (FC) above, we already predict the TWL impact of any topological node (gray in Fig. 4). Now, we propose a similar technique for rotational nodes (blue in Fig. 4). Whenever a die is added to the partial placement, it is likely connected to some terminals besides other dies. To account for terminals, we align each die individually in all four possible orientations to its terminals (Fig. 8(b)). Here we apply the analytical procedure (Sec. II-D) as well. We calculate and memorize the HPWL values for all respective die orientations.

We utilize these HPWL values analogous to forward checking. That is, for a rotational node, we sum up its TWL estimation and the rotation-specific minimal WL towards the respective terminals. Then, we compare this sum against the currently best solution's TWL. A node is allocated only for a promising configuration, and that node's estimated WL is set to the above sum. Unpromising branches are pruned.

E. Estimation of Remaining Distances

At this point, we know the *minimal augmentation* of the TWL *for each node* in the search tree. Now we can use this data to further estimate, at each level in the tree, how yet unassigned dies will enlarge the final TWL. We refer to these pending WL augmentations as *remaining distances*.

As each level in the search tree specifies either the rotation of dies or the topological relation between pairs of dies, the idea is to track the minimal TWL augmentation across all levels in the tree. To do so, we determine the most promising path in the tree across the minimal values over all levels. The remaining distance for a specific level is then simply the sum of minimal values along that path over all remaining (lower) levels, since these levels are responsible for yet unplaced dies.

Moreover, we can refine any remaining distances once we proceed with branching. For example, after assigning a new die to the search tree, we naturally know the orientation of this new die as well as the orientations of all previously assigned dies. Hence, the initial set of variations for forward checking (64) can be reduced to four on each level, and the remaining distances for the respective nodes can be recalculated more accurately.

Example of Estimation of Lower Bounds (Sec. III-C, D, E)

An example of the lower-bound estimation of the HPWL for partial configurations is given in Fig. 9. As always, the FC values are calculated here only for topological nodes. FC values represent the minimal wirelength between two dies for their specific rotations and their topology (recall Sec. III-C). The terminal handling (TH) values are only relevant for rotational nodes. TH values provide an estimation for the minimal HPWL between one die and its terminals (recall Sec. III-D). The remaining distances (RD) estimate the HPWL augmentation for yet unplaced dies for every level in the search tree (Sec. III-E).

The estimation procedure differs for defined and undefined partial configurations (Sec. II-B). For partial configurations, where topological relations between all dies are defined, we can construct the incomplete placement and calculate its $HPWL_{defined}$. The estimation for this node comprises then the $HPWL_{defined}$ and the estimated HPWL augmentation for yet unplaced dies (i.e., the RD value on the current level). Only in case the resulting value is smaller than that of the currently best solution, this partial configuration is considered as promising, and the related path in the search tree is continued. For undefined configurations (where we cannot derive a placement), the estimate for the node is calculated as the sum of: (i) the HPWL for the last defined configuration; (ii) all FC and TH values across the related path, from the last defined configuration to this node; and (iii) the RD value for this node's level.

F. Pruning Dominated Configurations

Extending the classical *dominance technique* for branch-and-bound algorithms, we seek to rule out configurations which appear much worse than others at the same level in the search tree. However, due to the non-linear nature of the HPWL metric (e.g., new pins can be inserted inside a net's bounding box without increasing the HPWL), partial configurations cannot be compared directly by their HPWL values. Thus, we propose a metric to select only the partial configurations that are assured to be unpromising/dominated. Nodes are discarded when they would increase the TWL by $(1/(dies\_number + 1) \times best\_TWL)$ beyond the estimated *minimal TWL increase* across all other nodes on the same level. Here, $dies\_number$ represents the total count of dies.

According to our experiments, pruning based on this metric allows to obtain the optimal solution for all considered benchmarks, along with notably speeding up the search.

G. Handling of Layout Constraints

We have this far identified unpromising partial configurations based on their WL, but they can also be identified based on geometrical and containment constraints (Section II). For complete solutions as well as for defined partial configurations, these constraints can be readily evaluated via their actual placements. However, undefined partial configurations cannot be transformed into a placement. Thus, we add topological constraints to the vertical and/or horizontal constraints graphs representing those undefined configurations. Then, we apply a topological sort to the graphs, which provides the minimal bounding box of all dies considered so far in this partial configuration, while also satisfying all given topological constraints. Since we apply such a topological sort for any new node anyway (to check whether its placement is feasible), these steps do not require any additional computational resources.

IV. Experimental Results

Our implementation was written in C++11 and compiled with gcc 4.9.3 (-O3 optimization flag). The runtime was evaluated on an Intel Xeon E5-2680 Linux workstation with 24 cores (disabled hyper-threading) running at 2.50 GHz.

We validated our algorithms empirically using three benchmark sets (Table I): (*i*) the interposer-oriented benchmarks from [16], which are a modified version of the ISPD08 benchmarks, (*ii*) the well-known floorplanning benchmarks MCNC, and (*iii*) modified MCNC variations. The MCNC benchmarks are used to compare our algorithms with prior art in optimal floorplanning [17, 18]. Additionally, we derive MCNC variants as follows: *xerox* with reduced die count (6–8 dies); *apte*, *xerox*, and *hp* with downscaled outlines, where the smaller whitespace makes the placement problem more practical. We publicly provide our tool as executable binary as well as our modified benchmarks [20].

A. Speed and Scalability of Our Approach

First, we compare two variants of our B&B implementation (on modified MCNC benchmarks with 6–8 dies, Table II): (*i*) native B&B and (*ii*) our advanced B&B with novel pruning techniques. We observe that the maximal number of dies that can be placed optimally for (i) is limited to six, which is consistent with previous studies [16, 17]. By applying our proposed techniques (ii), we can overcome this limit and significantly accelerate the search. Advanced pruning, for example, greatly reduces the number of nodes to be considered (from 263 million) to 350 thousand nodes for six dies. Moreover, recall that not all of these nodes have to be fully evaluated thanks to our techniques. The achieved speed-up for this test case is almost 2,000x. In general, whereas native B&B quickly reaches its limits and computations continue for several days, B&B using our techniques can solve the problem in a matter of seconds.

Figure 2 visualizes the search trees for a six-die benchmark, showing the effectiveness of our techniques. We observe that already for native B&B (Fig. 2(b)) many partial configurations are discarded, reducing the overall solution space. However, pruning typically makes a difference only at lower levels in the tree (when four or five dies have been placed). In contrast, our novel pruning techniques (Fig. 2(c)) are already effective once two dies are placed, thereby identifying unpromising partial configurations early on and accelerating the B&B search.

TABLE I
Benchmark characteristics

Bench		Dies	Pins	Nets	Terminals	Interposer	
						H(μm)	W(μm)
t4_s	mod. ISPD08	4	15,611	1,808	789	12,474	12,474
t4_m		4	91,005	5,326	1,174	25,542	25,707
t4_b		4	223,781	12,265	1,033	27,750	27,850
t6_s		6	20,138	1,720	639	16,324	16,324
t6_m		6	121,935	7,123	1,162	12,485	12,485
t6_b		6	299,228	14,264	1,192	23,250	23,400
t8_s		8	18,689	1,918	882	25,542	25,707
t8_m		8	159,149	8,391	1,391	20,592	20,724
t8_b		8	306,057	12,593	1,049	32,240	32,400
apte	MCNC	9	287	97	73	10,500	10,500
xerox		10	698	203	2	5,831	6,412
hp		11	309	83	45	4,928	4,200
xerox6	mod. MCNC	6	259	65	–	4,000	4,000
xerox7		7	329	85	–	4,500	4,500
xerox8		8	382	94	–	5,500	5,500
apte_s		9	287	97	73	see Table V	
xerox_s		10	698	203	2	see Table V	
hp_s		11	309	83	45	see Table V	

TABLE II
Comparison of the solution-space size and runtimes for different search algorithms. Also refer to Fig. 2 for xerox6

Bench	Brute-force	Native B&B		Advanced B&B with our pruning techniques	
	Nodes	Nodes ($\cdot 10^6$)	Time (s)	Nodes ($\cdot 10^6$)	Time (s)
xerox6	$4.398 \cdot 10^{12}$	263.2	585	0.351	0.25
xerox7	$7.205 \cdot 10^{16}$	–	>24h	18.198	13.1
xerox8	$4.722 \cdot 10^{21}$	–	>24h	786.425	684

B. Comparison with Prior Art

Next, we compare our results to those in previous studies: optimal die placement on the interposer in [16] (Table III) and optimal floorplanning in [17, 18] (Table IV).

TABLE III
COMPARISON BETWEEN OUR METHOD AND [16]

Bench	Enumeration from [16]				Bench	Our implementation		
	Optimal		Suboptimal					
	TWL* ($\cdot 10^6 \mu m$)	Time (s)	TWL* ($\cdot 10^6 \mu m$)	Time (s)		TWL* ($\cdot 10^6 \mu m$)	Time (s)	Speed-up
t4_s	10.43	0.49	10.44	0.72	t4_s*	10.87	0.2	x2.45–3.6
t4_m	36.24	4.59	36.26	8.63	t4_m*	38.14	0.58	x7.9–14.9
t4_b	63.27	17.48	63.27	19.67	t4_b*	58.92	1.16	x15.1–16.9
t6_s	8.84	958	8.84	2.87	t6_s*	9.01	0.45	x6.3–2,129
t6_m	30.51	3,528	30.52	6.88	t6_m*	33.77	8.56	x0.8–412.25
t6_b	55.23	1,607	55.24	17.31	t6_b*	62.71	4.74	x3.65–339
t8_s	26.27**	>12h	21.80**	11,289	t8_s*	23.51	3.21	x3,516–10^4
t8_m	39.05**	>12h	33.15**	11,146	t8_m*	36.39	159.1	x70–270
t8_b	69.06**	>12h	59.20**	11,351	t8_b*	66.61	20.46	x554.7–10^3

* The authors of [16] provided only benchmarks with re-generated positions for all terminals; their reported TWL values for original benchmarks and our TWL values obtained for re-generated benchmarks are not directly comparable.
** The results for large benchmarks reported for the optimal approach in [16] are worse than the suboptimal results since the authors interrupt the optimal search after 12h runtime.

TABLE IV
COMPARISON BETWEEN OUR METHOD AND [17, 18]

Bench	B&B [17], hierarchical mode*		CSP+B&B [18], no rotation of dies*		Our implementation	
	TWL (mm)	Time	TWL (mm)	Time	TWL (mm)	Time
apte	460	no info	513.06	13s	**437.51**	14m
xerox	566	38.5h	370.99	48s	**365.87****	>12h
hp	278	no info	153.33	102s	**150.26**	61m

* Both studies provide non-optimal results.
** Represents the best result obtained within twelve hours.

We observe that the enumeration-based tool of [16] can optimally place only six dies on the interposer and that the runtimes for this case are relatively high (15–60 minutes). Placing eight dies requires more than twelve hours. In contrast, we can find optimal placements for the six-die benchmarks in a few seconds, and for benchmarks with eight dies we require less than two and a half minutes. Liu *et al.* [16] also report suboptimal results, which they obtain in much shorter runtime than their optimal baseline technique. Still, even when comparing to their fast but suboptimal approach we obtain superior results in shorter runtime, especially when placing eight dies.

We also examine our solutions for the MCNC benchmarks with 9–11 dies (*apte, xerox, hp*), see Table IV and Fig. 10. No prior art that applies these benchmarks can provide optimal results: [17] is limited to six dies and leverages hierarchical placement for more dies, and [18] does not consider die rotations. These shortcomings are confirmed by our results (Table IV)—both studies report higher TWLs than ours. Although our tool runs longer than [18], we enable die rotations and thus ensure better results with accordingly "exhaustive" optimality.

The *xerox* benchmark (with 10 dies) is the most challenging for our approach, in terms of runtime and the number of nodes evaluated. We failed to cover the entire promising part of the solution space in twelve hours. Nonetheless and more importantly, we still obtain a better solution when compared to [18] within this allocated runtime. Interestingly, another benchmark with even one more die (*hp*, 11 dies) enforces much lower computational efforts. This indicates that there is not necessarily a direct correlation between benchmark complexity (number and shape of dies, pins, terminals, etc.) and runtime. We believe that our efforts depend more on (i) how many solutions are close to the optimum and (ii) the quality of lower-bound estimations for partial configurations. The larger the number of near-optimal solutions, and the lower the estimation for those nodes, the later they will be pruned—thereby inducing many nodes in the search tree to be evaluated.

TABLE V
RESULTS FOR DOWNSCALED MCNC BENCHMARKS

Bench	Scaled Dimenions H x W (μm x μm)	Whitespace* (%)	TWL (mm)	Time
apte_s	7630 x 7630	20	377.01	6s
apte_s	7400 x 7400	15	373.20	8s
apte_s	7000 x 7400	10	366.30	6s
apte_s	6400 x 7650	5	375.26	29s
xerox_s	5300 x 4530	20	363.99	3.6h
xerox_s	4550 x 5003	15	378.76	45m
xerox_s	4422 x 4862	10	419.98	34.8m
xerox_s	4380 x 4650	5	437.47	19.5m
hp_s	3600 x 3070	20	140.02	19s
hp_s	3492 x 2975	15	143.42	6s
hp_s	3350 x 2930	10	143.77	3s
hp_s	3310 x 2800	5	164.01	31m

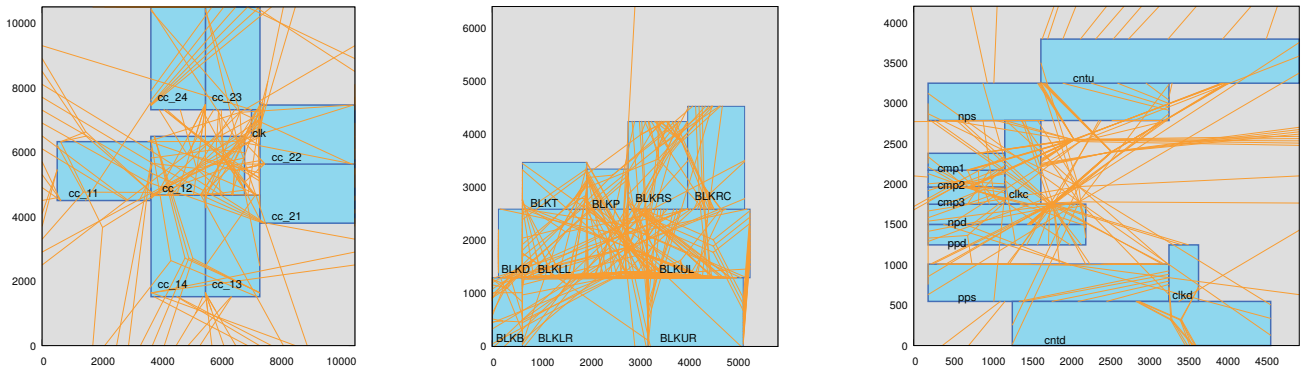
* For the original benchmarks, whitespace is as follows: 58% for apte, 48% for xerox, and 57% for hp.

C. Placement Trends for Practical Designs

Due to cost considerations, commercial interposer-based 3D ICs have typically only little whitespace available [10–12]. Thus, we also consider downscaled, modified versions of the MCNC benchmarks where the interposer area is reduced to allow for only up to 20% whitespace.

According to the results in Table V, benchmarks with more restricted placement areas are notably easier to place. Pruning due to containment constraints (recall Section III-G) plays a significant role here. Empirically, for the downscaled benchmarks we obtain best solutions within seconds or minutes for *apte* and *hp* benchmarks respectively, while *xerox* takes up to three and a half hours. This also reaffirms the benchmark-specific efforts observed in our prior experiments.

Another trend we observe is that the TWL increases after a certain point, which is due to the limited flexibility when placing dies onto downscaled interposers. For *hp* and *xerox*, the TWL degradation is already noticeable from 20% whitespace on; for *apte*, the threshold is about 10%. An increasing TWL, in turn, reduces the effectiveness of our WL-centric pruning techniques. Accordingly, we observe longer runtimes in some cases, e.g., at 5% whitespace for *hp*. Besides, we note that the aspect ratio affects the TWL significantly for downscaled interposers. For example, none of the designs could be placed when the original aspect ratio (AR) was applied and only 5% whitespace was provided. For 10% whitespace along with the original AR, *apte* and *hp* have been placed successfully, but with notable TWL overhead.



(a) (b) (c)

Fig. 10. Obtained placement for the original MCNC benchmarks: (a) apte, (b) xerox, and (c) hp. Yellow fly lines represent nets.

V. Summary

In this paper we tackled wirelength-driven and optimal die placement on interposers. Our method is based on several novel techniques that address the combinatorial explosion of the placement problem. Initially, we extended the meta-CSP approach from [19] to support die rotations. We applied this extension to mathematically represent placements and to construct the solution space. Further, we developed a parallel B&B algorithm to explore this solution space, and we enriched that algorithm with wirelength-centric pruning techniques to rule out unpromising configurations early on. These techniques include: a greedy, graph-based ordering of the search tree; wirelength-based forward checking before node allocation; estimation of minimal WL augmentations induced by temporarily yet unplaced dies; a fast (yet high-quality) determination of initial placement solutions; an extension of the B&B dominance concept; and early checking of containment constraints whether placements will eventually fit onto the interposer.

We have successfully validated our method using the ISPD08 and MCNC benchmarks. The results show a significant advancement over prior art: whereas previous works can determine optimal placement only for up to six dies of those benchmarks, we were able to optimally place up to eleven dies. Moreover, we outperformed all heuristic methods in solution quality.

For future work, we seek to address the trade-offs for TWL, whitespace quota and aspect ratio of the interposer. There are other interesting aspects worth investigating, such as optimal die placement on both sides of the interposer.

References

[1] A. B. Kahng, "Lithography-induced limits to scaling of design quality," in *SPIE Adv. Lithography*, vol. 9053, 2014.

[2] R. S. Patti, "Three-dimensional integrated circuits and the future of system-on-chip designs," *Proc. IEEE*, vol. 94, no. 6, pp. 1214–1224, 2006.

[3] J. H. Lau, "The most cost-effective integrator (TSV interposer) for 3D IC integration system-in-package (SiP)," in *Proc. ASME InterPACK*, 2011, pp. 53–63.

[4] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *Proc. Int. Symp. Microarch.*, 2015, pp. 546–558.

[5] P. Batude *et al.*, "Advances, challenges and opportunities in 3D CMOS sequential integration," in *Proc. Int. Elec. Devices Meeting*, 2011, pp. 7.3.1–7.3.4.

[6] D. Stow, I. Akgun, R. Barnes, P. Gu, and Y. Xie, "Cost analysis and cost-driven IP reuse methodology for SoC design based on 2.5D/3D integration," in *Proc. Int. Conf. Comp.-Aided Des.*, 2016, pp. 56:1–56:6.

[7] T. Lu, C. Serafy, Z. Yang, S. Samal, S. K. Lim, and A. Srivastava, "TSV-based 3D ICs: Design methods and tools," *Trans. Comp.-Aided Des. Integ. Circ. Sys.*, vol. 36, no. 10, pp. 1593–1619, 2017.

[8] S. Osmolovskyi and J. Lienig, "Physical design challenges and solutions for interposer-based 3D systems," in *Reliability by Design; 9. ITG/GMM/GI-Symposium*, 2017, pp. 1–8.

[9] J. Knechtel, O. Sinanoglu, I. A. M. Elfadel, J. Lienig, and C. C. N. Sze, "Large-scale 3D chips: Challenges and solutions for design automation, testing, and trustworthy integration," *Trans. Sys. LSI Des. Method.*, vol. 10, pp. 45–62, 2017. [Online]. Available: http://doi.org/10.2197/ipsjtsldm.10.45

[10] R. Chaware, K. Nagarajan, and S. Ramalingam, "Assembly and reliability challenges in 3D integration of 28nm FPGA die on a large high density 65nm passive interposer," in *Proc. Elec. Compon. Technol. Conf.*, 2012, pp. 279–283.

[11] J. Macri, "AMD's next generation GPU and high bandwidth memory architecture: FURY," in *Hot Chips Symp.*, 2015, pp. 1–26.

[12] P. Dorsey, "Xilinx stacked silicon interconnect technology delivers breakthrough FPGA capacity, bandwidth, and power efficiency," Xilinc, Inc., Tech. Rep., 2010.

[13] B. Martin, K. Han, and M. Swaminathan, "A path finding based SI design methodology for 3D integration," in *Proc. Elec. Compon. Tech. Conf.*, 2014, pp. 2124–2130.

[14] Y.-K. Ho and Y.-W. Chang, "Multiple chip planning for chip-interposer codesign," in *Proc. Des. Autom. Conf.*, 2013, pp. 27:1–27:6.

[15] D. Seemuth, A. Davoodi, and K. Morrow, "Automatic die placement and flexible I/O assignment in 2.5D IC design," in *Proc. Int. Symp. Quality Elec. Des.*, 2015, pp. 524–527.

[16] W.-H. Liu, M.-S. Chang, and T.-C. Wang, "Floorplanning and signal assignment for silicon interposer-based 3D ICs," in *Proc. Des. Autom. Conf.*, 2014, pp. 5:1–5:6.

[17] H. Onodera, Y. Taniguchi, and K. Tamaru, "Branch-and-bound placement for building block layout," *El. and Commun. in Japan (Part III: Fund. El. Science)*, vol. 76, no. 7, pp. 15–26, Jan. 1993.

[18] J. Funke, S. Hougardy, and J. Schneider, "An exact algorithm for wirelength optimal placements in VLSI design," *Integration*, vol. 52, pp. 355–366, 2016.

[19] R. E. Korf, M. D. Moffitt, and M. E. Pollack, "Optimal rectangle packing," *Annals of Oper. Research*, vol. 179, no. 1, pp. 261–295, Nov. 2010.

[20] S. Osmolovskyi, "Placement framework for interposer-based 3D ICs," 2017. [Online]. Available: http://www.ifte.de/english/research/interposer-design/index.html

[21] X. Tang, R. Tian, and M. D. F. Wong, "Minimizing wire length in floorplanning," *Trans. Comput.-Aided Des. Integr. Circuits Sys.*, vol. 25, no. 9, pp. 1744–1753, 2006.