VALIDITY-PRESERVING HIERARCHICAL RL FOR JOINT ROUTING AND SWITCH PLACEMENT IN EDA

Anonymous authors
Paper under double-blind review

ABSTRACT

Routing and switch placement are fundamental combinatorial optimization problems in chip design, requiring the joint optimization of routing topology and physical placement under strict structural, geometric and logical constraints. Existing approaches typically rely on carefully engineered heuristics that incorporate strong problem-specific biases to navigate the enormous space of possible designs. In this work, we introduce a hierarchical reinforcement learning framework for joint routing and switch placement at the level of logical communication routes. Starting from a minimal routing graph, our method progressively constructs increasingly expressive solutions through three coupled operations: *switch expansion*, *switch placement*, and *route refinement*. These operations preserve routing validity by construction, restricting exploration to feasible configurations where every communicating initiator-target pair has one assigned loop-free route. We explore the induced solution space using Gumbel Monte Carlo Tree Search, showing that neural-guided search substantially improves solution quality over non-learning optimization methods. Furthermore, pretraining across floorplans provides a strong initialization for fine-tuning on unseen instances. Our code will be made publicly available upon acceptance.

1 INTRODUCTION

Routing and placement are central optimization problems in modern chip design. Given a set of communicating components and their physical environment, routing must establish connections while optimizing objectives such as wirelength and congestion under physical and topological constraints (Marculescu et al., 2008; Hu & Sapatnekar, 2001). These problems are combinatorial: the routing structure, the switch locations, and the individual communication routes interact, and even restricted Steiner-type formulations arising in Very-Large-Scale Integration (VLSI) design are computationally hard (Tang et al., 2020; Ihler et al., 1999).

Classical approaches address the complexity stemming from Electronic Design Automation (EDA) through carefully designed combinatorial optimization procedures and domain-specific heuristics (Chu & Wong, 2005; Hu & Sapatnekar, 2001; Lee, 1961). Learning-based methods, and Reinforcement Learning (RL) in particular, provide an alternative in which the strategy used to explore a combinatorial design space can itself be learned. Recent successes in placement and routing demonstrate the potential of this approach for chip design (Mirhoseini et al., 2021; Liu et al., 2021; Chen et al., 2022; Du et al., 2023; Liu et al., 2024; Du et al., 2026).

Routing and switch placement are tightly coupled: *switch locations determine which routing structures are effective, while the routing structure determines which switch locations are useful*. Optimizing these decisions sequentially can therefore sacrifice solution quality by committing to decisions before accounting for their effect on the other problem. We instead optimize routing and switch placement jointly. We deliberately study a simplified physical model that captures the interaction between shared routing topology, switch placement, and communication routes while abstracting away constraints such as routing layers, capacities, vias, and detailed design rules. We view this formulation as a first step toward an optimization core that can subsequently be extended with richer physical constraints and cost models.

In this work, we introduce a hierarchical construction process that progressively builds solutions through three operations: *i) switch expansion*, which introduces a new switch; *ii) switch placement*,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

which assigns its physical location; and *iii) route refinement*, which updates the routes affected by the expansion. The resulting hierarchy restricts exploration to a subset of feasible solutions while ensuring that a globally optimal solution remains reachable, thereby reducing the search space without prescribing how it should be explored. We formulate this construction process as a sequential decision problem and learn graph policies to guide exploration using Gumbel Monte Carlo Tree Search (MCTS) (Danihelka et al., 2022). Finally, we train a policy jointly across floorplans and study whether the resulting initialization can accelerate optimization on previously unseen instances. Our contributions are as follows:

- We introduce a hierarchical graph formulation for joint routing and switch placement that restricts search to feasible configurations while ensuring that a globally optimal solution remains reachable (Section 4).
- We combine this formulation with a learned graph policy and Gumbel MCTS, and show that the resulting search can exploit additional compute to progressively improve solution quality and outperform the evaluated non-learning baselines (Section 5 and Table 1).
- We demonstrate that learned priors can be transferred across floorplans: a policy trained jointly on multiple instances provides a transferable search prior that accelerates optimization when fine-tuned on previously unseen floorplans (Figure 5).

2 Related Work

Classical methods. Classical physical-design methods rely on optimized combinatorial and continuous optimization procedures. Analytical placement methods such as RePlAce (Cheng et al., 2018) optimize differentiable placement objectives under density constraints, while physical routing commonly relies on Steiner-tree construction (Tang et al., 2020; Hu & Sapatnekar, 2001; Chu & Wong, 2005), shortest-path and maze-routing procedures (Lee, 1961), and iterative rip-up-and-reroute (Dees & Smith, 1981; Linsker, 1984; Kahng et al., 2021). Application-specific Network-on-Chip (NoC) synthesis similarly considers the joint design of communication architectures for a given application. Ahonen et al. (2004) optimize application-specific network topologies using communication requirements and physical wirelength information, while Morgan et al. (2013) jointly optimize topology selection, core mapping, and traffic routing through a multi-objective genetic algorithm. More broadly, classical NoC synthesis methods rely on combinatorial optimization and domain-specific heuristics to navigate large design spaces (Marculescu et al., 2008).

Learning-based methods. Machine learning has been applied to several physical-design problems. AlphaChip (Mirhoseini et al., 2021) formulates macro placement as a sequential decision problem and uses RL to optimize placement quality. Subsequent works have incorporated routing information into learning-based physical-design pipelines. Cheng & Yan (2021) use routing results to evaluate placement quality, while Cheng et al. (2022) combine RL-based placement with a conditional generative routing model. Other approaches have explored black-box optimization for macro placement (Shi et al., 2023) and combinations of RL and tree search (Geng et al., 2024).

Learning has also been applied directly to routing and Steiner-tree construction. Liao et al. (2020) formulate physical routing as a sequential RL problem, while REST (Liu et al., 2021) uses RL to construct rectilinear Steiner minimum trees. For obstacle-aware routing, Chen et al. (2022) combine RL with MCTS for Steiner-point selection. Other approaches learn intermediate geometric structures: HubRouter (Du et al., 2023) generates hubs that guide pin-hub connections, while NeuralSteiner (Liu et al., 2024) predicts candidate Steiner points. More recently, OAREST (Du et al., 2026) uses RL for obstacle-avoiding rectilinear Steiner minimum tree construction and introduces a restricted representation shown to retain an optimal solution.

Our setting differs from these Steiner-tree formulations by optimizing a shared communication infrastructure for multiple communication pairs rather than a tree connecting the pins of a single net. Our formulation jointly optimizes the topology and placement of intermediate switches and the route assigned to each communication pair, allowing communications to share physical infrastructure. It is also related to application-specific NoC synthesis, but focuses on a simplified geometric setting with fixed communicating components and directly constructs the intermediate infrastructure. Within this setting, our hierarchical construction provides a structured search space that preserves feasible assigned routes and contains a globally optimal solution under our model.

3 PROBLEM FORMULATION

Notations. We use calligraphic letters (*e.g.*, $\mathcal{V}$) for sets, with cardinality $|\mathcal{V}|$. Bold lowercase letters denote vectors (*e.g.*, $\mathbf{p}$). For a point $\mathbf{p} \in \mathbb{R}^2$, we write $\mathbf{p} = [x, y]$ for its horizontal and vertical coordinates.

Basic definitions. A directed graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of directed edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. An edge $(u, v) \in \mathcal{E}$ is directed from u to v. A directed path from v_0 to v_k is a sequence of vertices $\pi = (v_0, v_1, \ldots, v_k)$ such that $(v_j, v_{j+1}) \in \mathcal{E}$ for all $j \in \{0, \ldots, k-1\}$. We denote by $E(\pi) = \{(v_j, v_{j+1}) : 0 \le j < k\}$ the set of edges traversed by the directed path π. A path is *simple* if it contains no repeated vertex, *i.e.*, $v_j \neq v_\ell$ for all $j \neq \ell$.

3.1 PROBLEM SETTING

We consider the joint optimization of routing and switch placement for a fixed set of communicating components. Let $\mathcal{I}$ and $\mathcal{T}$ denote the sets of initiators and targets, representing the source and destination endpoints of communication requests, respectively, such as processing, memory, or other IP blocks. Their positions are fixed within a rectangular floorplan $\Omega = [0, W] \times [0, H] \subset \mathbb{R}^2$. The communication requirements are specified by a set $\mathcal{R} \subseteq \mathcal{I} \times \mathcal{T}$ of communication pairs, *i.e.*, initiator-target pairs for which a communication route must exist. We additionally consider a set of rectangular blockages $\mathcal{B} \subset \Omega$, whose interiors cannot contain switches or be traversed by wires.

Given these inputs, we jointly determine the number and positions of intermediate switches, the routing structure connecting them, and the route assigned to each communication pair. Let $\mathcal{S}$ denote the set of switches, with $|\mathcal{S}| \le S_{\max}$, and let $\mathbf{p}_v = [x_v, y_v] \in \Omega \setminus \mathcal{B}$ denote the position of each switch $v \in \mathcal{S}$. Together with the fixed initiators and targets, these switches define a directed routing graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{I} \cup \mathcal{T} \cup \mathcal{S}$. For each communication pair $(i, t) \in \mathcal{R}$, the routing solution assigns a simple directed path $\pi_{i,t} = (v_0, \ldots, v_k)$, $v_0 = i$, $v_k = t$, that traverses at least one switch, with all intermediate nodes belonging to $\mathcal{S}$. The routing graph is induced by these paths, such that $\mathcal{E} = \bigcup_{(i,t)\in\mathcal{R}} E(\pi_{i,t})$. We make several simplifying assumptions: initiator, target, and switch dimensions and pin-level constraints are ignored, and routing density, congestion, and bandwidth constraints are not modeled.

3.2 OBJECTIVE

We consider rectilinear routing, where wires are composed exclusively of horizontal and vertical segments. For two nodes u, v with positions $\mathbf{p}_u$ and $\mathbf{p}_v$, let $\Gamma_{\mathcal{B}}(u, v)$ denote the set of rectilinear paths between $\mathbf{p}_u$ and $\mathbf{p}_v$ that do not intersect the interior of any blockage. We define the obstacle-avoiding rectilinear distance as $d_{\mathcal{B}}(u, v) = \min_{\gamma\in\Gamma_{\mathcal{B}}(u,v)} \mathrm{len}(\gamma)$. When an unobstructed Manhattan-shortest path exists, this reduces to $d_{\mathcal{B}}(u, v) = |x_u - x_v| + |y_u - y_v|$.

Both the physical extent of the interconnect and the lengths of individual communication paths are important considerations in on-chip network design (Ahonen et al., 2004; Morgan et al., 2013). We therefore distinguish between the total physical wirelength of the routing graph and the route lengths of individual communications. The total wirelength is $L_{\mathrm{wire}} = \sum_{\{u,v\}:\,(u,v)\in\mathcal{E}\,\vee\,(v,u)\in\mathcal{E}} d_{\mathcal{B}}(u, v)$, where each physical connection is counted once, regardless of its traversal direction or the number of communication pairs using it. For a route $\pi_{i,t} = (v_0 = i, \ldots, v_k = t)$, its length is $\mathrm{len}(\pi_{i,t}) = \sum_{j=0}^{k-1} d_{\mathcal{B}}(v_j, v_{j+1})$, and the total route length is $L_{\mathrm{route}} = \sum_{(i,t)\in\mathcal{R}} \mathrm{len}(\pi_{i,t})$. The wirelength encourages compact and shared wires, while the route length discourages long communication paths between individual initiator–target pairs. Let $\Pi_{i,t}(\mathcal{S})$ denote the set of simple directed paths from i to t that traverse at least one switch and whose intermediate vertices belong to $\mathcal{S}$, *i.e.*, $\Pi_{i,t}(\mathcal{S}) = \{\pi_{i,t} = (v_0, \ldots, v_k) \mid v_0 = i,\ v_k = t,\ k \ge 2,\ v_j \in \mathcal{S}\ \forall j \in \{1, \ldots, k-1\},\ v_j \neq v_\ell\ \forall j \neq \ell\}$: the joint routing and switch-placement problem is defined as

$$\min_{(\mathcal{S},\mathbf{p},\pi)\in\mathcal{F}} L_{\mathrm{wire}} + \lambda L_{\mathrm{route}}, \qquad \mathcal{F} = \left\{ \begin{array}{l|l} \big(\mathcal{S}, \{\mathbf{p}_v\}_{v\in\mathcal{S}}, \{\pi_{i,t}\}_{(i,t)\in\mathcal{R}}\big) & \mathbf{p}_v \in \Omega \setminus \mathcal{B}\ \forall v \in \mathcal{S} \\ |\mathcal{S}| \le S_{\max}; \quad \pi_{i,t} \in \Pi_{i,t}(\mathcal{S})\ \forall (i,t) \in \mathcal{R} & \end{array} \right\}. \tag{1}$$

In our experiments, we set $\lambda = \frac{1}{2}$. The problem is closely related to the rectilinear Steiner tree problem, which is NP-hard (Garey & Johnson, 1977).

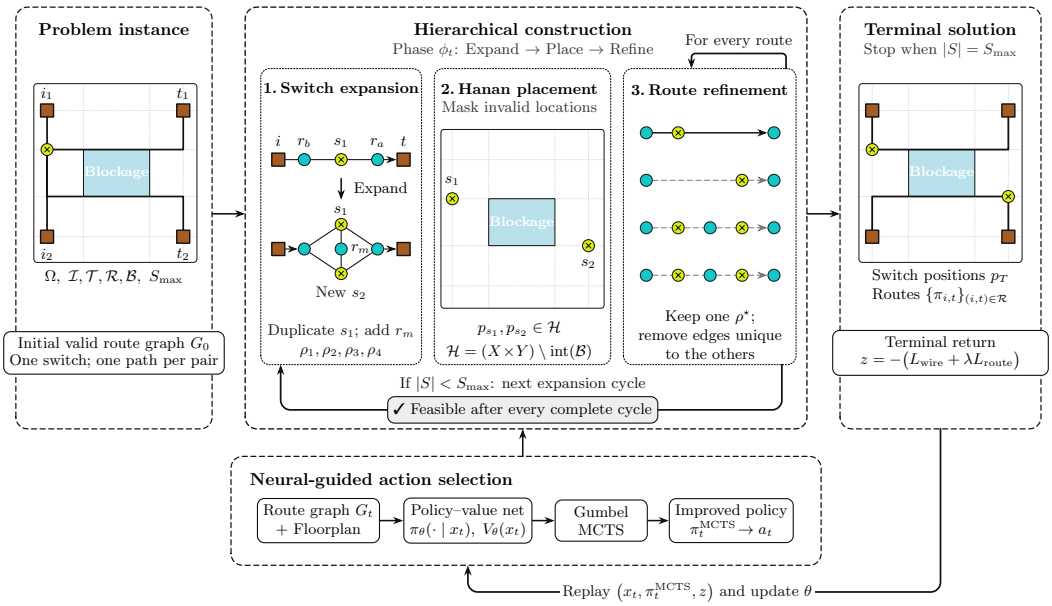


Figure 1: Overview of our routing and switch-placement method. Gumbel MCTS guides repeated switch expansion, Hanan-grid placement, and route refinement while preserving routing feasibility.

4 METHOD

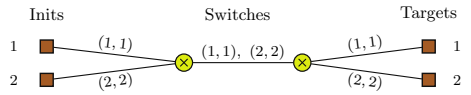
Overview. We formulate joint routing and switch placement as an iterative construction process over a graph representation. Starting from a minimal feasible routing solution containing a single switch, the routing graph is progressively expanded by introducing new switches, placing them on an extended Hanan grid (Hanan, 1966), and updating the routes affected by each introduction. Restricting switch placement to this finite set of candidate locations does not exclude a globally optimal solution, but replaces the continuous placement space with a finite one. Each complete refinement step returns a feasible routing configuration, so that search is restricted to valid solutions rather than arbitrary routing graphs. This process is repeated up to a predefined switch budget. We formulate these decisions as a Markov Decision Process and learn a graph policy and value function using Gumbel MCTS (Danihelka et al., 2022), with solution quality evaluated according to the objective in Equation (1). Figure 1 summarizes our proposed method. Complete proofs of all propositions are provided in Appendix A.

4.1 ROUTE-NODE REPRESENTATION

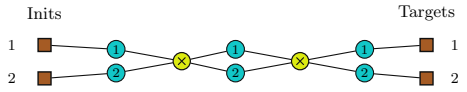
The routing graph defined in Section 3 specifies the physical connectivity induced by the communication routes. However, an edge may be shared by several communication pairs. Representing these assignments as variable-size edge attributes is inconvenient for our construction process. We therefore make individual route assignments explicit in the graph by introducing *route nodes*.

Definition 4.1 (Route-node conversion)**.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a routing graph induced by a collection of communication routes. For each edge $(u, v) \in \mathcal{E}$, let $\mathcal{R}(u, v) = \{(i, t) \in \mathcal{R} : (u, v) \in E(\pi_{i,t})\}$ denote the communication pairs whose routes traverse that edge. We replace (u, v) by one two-edge path $u \to r_{i,t}^{u,v} \to v$ for each $(i, t) \in \mathcal{R}(u, v)$, where $r_{i,t}^{u,v}$ is a route node associated with communication pair (i, t).

The resulting representation makes each use of a physical connection by a communication pair explicit. Route assignments can therefore be modified through local graph operations rather than variable-size edge attributes. Figure 2 illustrates the conversion. The transformed graph is bipartite between physical nodes (initiators, targets, and switches) and route nodes. Each route node has exactly one incoming and one outgoing edge and is associated with a single communication pair. Moreover, for every communication pair traversing a switch, an incoming route node is paired with



(a) Routing graph with communication pairs stored as edge attributes.



(b) Graph after conversion to route nodes. Route nodes labeled *(1)* correspond to pair $(1, 1)$ and those labeled *(2)* to pair $(2, 2)$.

Figure 2: Route-node conversion. Each communication pair traversing an edge is represented explicitly by a route node.

an outgoing route node of the same communication pair. These structural properties are invariants preserved by the construction operations introduced below.

4.2 Candidate Switch Locations

Switch positions are initially continuous variables over the floorplan. To obtain a finite placement space, we restrict candidate locations to an extended *Hanan grid*, a classical construction for rectilinear routing (Hanan, 1966). Let $\mathcal{P}$ denote the positions of all initiators and targets, and let $\mathcal{C}_{\mathcal{B}}$ denote the corners of the rectangular blockages. We define the sets of horizontal and vertical coordinates $\mathcal{X} = \{x(p) : p \in \mathcal{P}\} \cup \{x(c) : c \in \mathcal{C}_{\mathcal{B}}\}$, $\mathcal{Y} = \{y(p) : p \in \mathcal{P}\} \cup \{y(c) : c \in \mathcal{C}_{\mathcal{B}}\}$. The set of candidate switch locations is then

$$\mathcal{H} = \{(x, y) \in \mathcal{X} \times \mathcal{Y} : (x, y) \notin \mathrm{int}(\mathcal{B})\}. \tag{2}$$

Figure 3 illustrates the resulting grid.

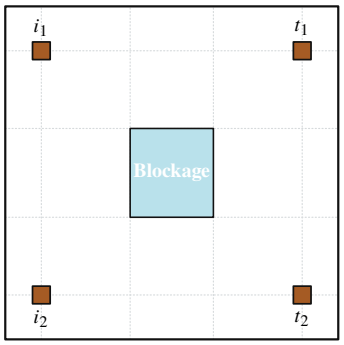


Figure 3: Extended Hanan grid induced by initiators, targets, and blockage corners.

We have the following result :

Proposition 4.2 (Optimal Switch Placement)**.** *Under rectilinear routing, and for fixed positions of initiators, targets, and blockages, there exists an optimal solution in which every switch is placed at an intersection of the extended Hanan grid $\mathcal{H}$.*

Thus, restricting switch placement to $\mathcal{H}$ preserves an optimal solution while reducing the continuous placement problem to a finite set of candidate locations. Therefore, in Equation (1), we can replace the search space $\mathbf{p}_v \in \Omega \setminus \mathcal{B},\ \forall v \in \mathcal{S}$ by $\mathbf{p}_v \in \mathcal{H},\ \forall v \in \mathcal{S}$.

4.3 Switch Expansion and Route Refinement

In this section, we describe how the routing graph is locally expanded and refined. Given a switch s_1, switch expansion introduces a new switch s_2 and temporarily introduces routing alternatives associated with s_1 and s_2. Route refinement subsequently resolves these alternatives independently for each affected communication pair. Those two operations are formally defined as follows.

Definition 4.3 (Switch expansion)**.** Let s_1 be the switch selected for expansion, and let $\mathcal{N}_b(s_1)$ and $\mathcal{N}_a(s_1)$ denote the sets of route nodes immediately preceding and following s_1, respectively. Switch expansion introduces a new switch s_2. For every $r_b \in \mathcal{N}_b(s_1)$, we add the edge $r_b \to s_2$, and for every $r_a \in \mathcal{N}_a(s_1)$, we add the edge $s_2 \to r_a$.

For each communication pair (i, t) traversing s_1, we additionally introduce a route node $r_m^{i,t}$ associated with (i, t) and add the edges $s_1 \to r_m^{i,t} \to s_2,\ s_2 \to r_m^{i,t} \to s_1$.

Remark 4.4. Informally, the expansion duplicates the selected switch and introduces, for each affected communication pair, an additional route node between the two switches. If r_b and r_a denote the route nodes immediately before and after the expanded switch, this exposes four local routing alternatives:

$$\rho_1 : r_b \to s_1 \to r_a, \qquad \rho_2 : r_b \to s_2 \to r_a,$$
$$\rho_3 : r_b \to s_1 \to r_m \to s_2 \to r_a, \quad \rho_4 : r_b \to s_2 \to r_m \to s_1 \to r_a.$$

Route refinement subsequently selects one of these four alternatives.

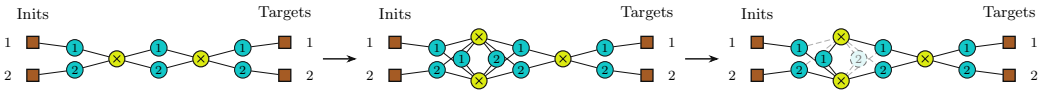


Figure 4: Switch expansion and route refinement. Expansion of a switch (leftmost yellow node in the first part of the figure) introduces a second switch and exposes four local routing alternatives for each affected communication pair. Route refinement selects one alternative, restoring a unique local route while leaving the remainder of the communication path unchanged.

Definition 4.5 (Route refinement)**.** For each communication pair affected by the expansion, let $\{\rho_1, \rho_2, \rho_3, \rho_4\}$ denote its four local routing alternatives, where each ρ_j is identified with its set of edges. Given a selected alternative $\rho^\star$, route refinement removes all edges belonging exclusively to the unselected alternatives: $\mathcal{E} \leftarrow \mathcal{E} \setminus \left(\bigcup_{\rho_j \neq \rho^\star} E(\rho_j) \setminus E(\rho^\star) \right)$. Route nodes left disconnected by this operation are removed.

Figure 4 illustrates the expansion and subsequent refinement process. Although the expanded intermediate graph contains multiple alternatives for the affected communication pairs, refinement restores a feasible routing configuration.

Proposition 4.6 (Validity preservation)**.** *Assume the routing configuration before expansion assigns exactly one simple directed path to every communication pair. After applying switch expansion and route refinement (Definitions 4.3 and 4.5), the resulting routing configuration again assigns exactly one simple directed path to every communication pair.*

This proposition states that the feasibility of the constructed routing is an invariant of the expansion–refinement cycle, *i.e.*, starting from a minimal feasible routing with a single switch, any routing constructed using those two operations remains feasible. However, the hierarchical construction cannot generate every feasible routing configuration. Indeed, consider the following example:

Example 4.7. Consider three switches s_1, s_2, s_3 and three communication routes $\pi_1 = (s_1, s_2, s_3)$, $\pi_2 = (s_2, s_3, s_1)$, $\pi_3 = (s_3, s_1, s_2)$. This configuration cannot be generated by the hierarchical construction following Definitions 4.3 and 4.5. Indeed, consider the switch introduced last and assume, without loss of generality, that it is s_3. This switch must be introduced by expanding either s_1 or s_2. For any route containing both the parent switch and s_3, these two switches are consecutive immediately after the expansion. Since s_3 is introduced last, no subsequent expansion can insert another switch between them. If s_3 is introduced by expanding s_1, this contradicts $\pi_1 = (s_1, s_2, s_3)$. If it is introduced by expanding s_2, this contradicts $\pi_3 = (s_3, s_1, s_2)$. Hence the configuration cannot be generated by the hierarchical construction.

Nevertheless, the following result shows that excluding such configurations does not sacrifice global optimality: at least one globally optimal solution always belongs to the hierarchical search space.

Proposition 4.8 (Optimality of the hierarchical search space)**.** *There exists a globally optimal solution to Equation* (1) *that can be obtained from the initial configuration through a finite sequence of switch expansions, placements, and route refinements.*

4.4 Routing as a Sequential Markov Decision Process

We formulate the hierarchical construction process as a Markov Decision Process (MDP) $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. An episode starts from a single switch connecting every communication pair and progressively refines an initially feasible routing configuration through successive switch expansions, switch placements, and route refinements.

State space. At step t, the state is defined as $x_t = \left(G_t, \mathbf{p}_t, \phi_t, \mathcal{Q}_t^{\text{place}}, \mathcal{Q}_t^{\text{route}} \right)$, where $G_t = (\mathcal{V}_t, \mathcal{E}_t)$ is the current route-node graph, $\mathbf{p}_t = \{\mathbf{p}_v\}_{v \in \mathcal{S}_t}$ contains the positions of the current switches, and ϕ_t denotes the current decision phase. The queues $\mathcal{Q}_t^{\text{place}}$ and $\mathcal{Q}_t^{\text{route}}$ contain, respectively, the switches awaiting placement and the communication routes awaiting refinement. The next placement or refinement decision always operates on the first element of the corresponding queue.

Initial state. The initial graph G_0 contains a single switch $s^{(0)}$. For every communication pair $(i,t) \in \mathcal{R}$, two route nodes $r_{i,t}^{\text{in}}$ and $r_{i,t}^{\text{out}}$ form the path $i \to r_{i,t}^{\text{in}} \to s^{(0)} \to r_{i,t}^{\text{out}} \to t$. The initial switch is placed at a fixed valid location $\mathbf{p}_{s^{(0)}} \in \mathcal{H}$. Both queues are initially empty, $\mathcal{Q}_0^{\text{place}} = \mathcal{Q}_0^{\text{route}} = \emptyset$, and the initial phase corresponds to switch expansion. The initial state therefore represents a feasible routing solution.

Action space. The set of admissible actions depends on the current phase ϕ_t. During switch expansion, the action selects a switch $s \in \mathcal{S}_t$ to expand according to Definition 4.3. During switch placement, the action selects a valid candidate location $\mathbf{p} \in \mathcal{H}$ for the first switch in $\mathcal{Q}_t^{\text{place}}$. During route refinement, the action selects one of the four local routing alternatives $\rho^\star \in \{\rho_1, \rho_2, \rho_3, \rho_4\}$ for the first route in $\mathcal{Q}_t^{\text{route}}$.

Transition dynamics. The environment is deterministic and Markovian. Each construction cycle begins with an expansion action, which introduces a new switch that initially inherits the position of the expanded switch. The two switches affected by the expansion are added to $\mathcal{Q}_t^{\text{place}}$, and the affected communication routes are added to $\mathcal{Q}_t^{\text{route}}$. The placement and refinement queues are then processed sequentially. Each action operates on and removes the first element of the corresponding queue. Once $\mathcal{Q}_t^{\text{place}}$ is empty, the process transitions from placement to route refinement. Once $\mathcal{Q}_t^{\text{route}}$ is also empty, the resulting graph again represents a feasible routing configuration and the process returns to the switch expansion phase. The episode terminates when the switch budget $S_{\max}$ is reached and the final refinement cycle has been completed.

Reward. We use a sparse terminal reward corresponding to the negative routing objective of Equation (1): $r_t = 0$ for all $t < T$, and $r_T = -L_{\text{wire}}(G_T) - \lambda L_{\text{route}}(G_T)$, where $\lambda = \frac{1}{2}$ in our experiments.

5 Implementation

We learn a shared graph policy and value network over the sequential decision process defined in Section 4.4. The network jointly encodes the current routing graph, blockage geometry, and candidate switch locations with separate policy heads for switch expansion, placement, and route refinement. Invalid actions are masked throughout construction. We use Gumbel MCTS (Danihelka et al., 2022) to explore the resulting search space, using the learned policy and value function to guide tree search and training them toward search-improved targets. Following the risk-seeking value estimation of AlphaTensor (Fawzi et al., 2022), we train the value function toward the top 25% of observed returns rather than their average. Per-floorplan PopArt (Hessel et al., 2019) is used for value normalization.

To isolate the contribution of tree search, we additionally train PPO-EWMA (Hilton et al., 2022) on the same hierarchical MDP. PPO-EWMA uses the same policy-value architecture, action masking, terminal objective, and PopArt value normalization, but selects actions directly from the learned policy without tree search. Architectural details, optimization procedures, and hyperparameters are provided in Appendix C.

6 Experiments and Results

Our experiments investigate whether (*i*) learned guidance improves over non-learning optimization methods, (*ii*) explicit tree search improves over direct policy optimization, and (*iii*) pretraining across floorplans improves optimization on unseen instances. Experimental details, numerical results, and routing visualizations are provided in Appendices C, D, and E, respectively.

6.1 Datasets and Experimental Setup

Floorplans. We evaluate our model on 28 synthetic square floorplans with rectangular blockages and initiator–target communication constraints. We use 24 floorplans for pretraining and hold out the remaining four for transfer experiments. The instances contain 18–25 communication pairs, 3–5 initiators, and 5–8 targets, with switch budgets ranging from 2 to 5. The unobstructed area covers 58.4%–81.3% of each floorplan. We construct an extended Hanan grid from terminal coordinates and blockage boundaries, yielding grids ranging from 24×23 to 42×44 candidate positions.

Table 1: Objective values on the 24 training floorplans (lower is better). Best results are shown in bold and second-best results are underlined.

Method	1	2	3	4	5	6	7	8	9	10	11	12
Heuristic	11.902	11.255	15.349	11.330	14.847	11.126	8.090	10.301	11.292	14.615	14.741	13.418
Random search	18.706	20.799	24.042	17.420	23.629	19.310	12.672	18.574	18.663	24.931	22.584	20.775
Genetic algorithm	14.382	13.999	16.854	12.622	17.970	13.107	8.224	11.954	14.152	15.799	16.478	16.504
PPO-EWMA	12.166	11.484	15.670	11.576	14.717	10.960	8.595	10.352	11.142	13.329	14.151	**13.361**
Gumbel MCTS	**11.333**	**10.880**	**14.274**	**9.784**	**13.926**	**9.918**	**7.739**	**9.897**	**10.820**	**13.028**	**13.995**	**13.361**

Method	13	14	15	16	17	18	19	20	21	22	23	24
Heuristic	13.662	15.003	14.512	15.284	16.571	16.366	16.277	16.277	14.909	15.415	14.007	15.684
Random search	21.425	21.875	19.338	20.146	20.943	30.810	29.030	20.421	23.683	21.337	22.457	20.583
Genetic algorithm	15.238	15.057	**14.134**	**15.236**	**15.239**	19.881	22.062	**15.726**	16.580	14.531	14.621	**15.277**
PPO-EWMA	**13.287**	13.593	**14.134**	15.269	**15.239**	19.147	18.488	15.817	13.095	13.989	12.409	**15.277**
Gumbel MCTS	**13.287**	**13.466**	**14.134**	**15.236**	**15.239**	**13.601**	**14.373**	**15.726**	**12.779**	**13.471**	**12.386**	**15.277**

Metric. We evaluate solutions using the objective defined in Section 3, *i.e.*, $L_{\text{wire}} + \frac{1}{2}L_{\text{route}}$. Wirelength counts each physical connection once, irrespective of the number or direction of routes using it, while route length counts every route traversal. For readability, all lengths are normalized by the side length of the corresponding floorplan.

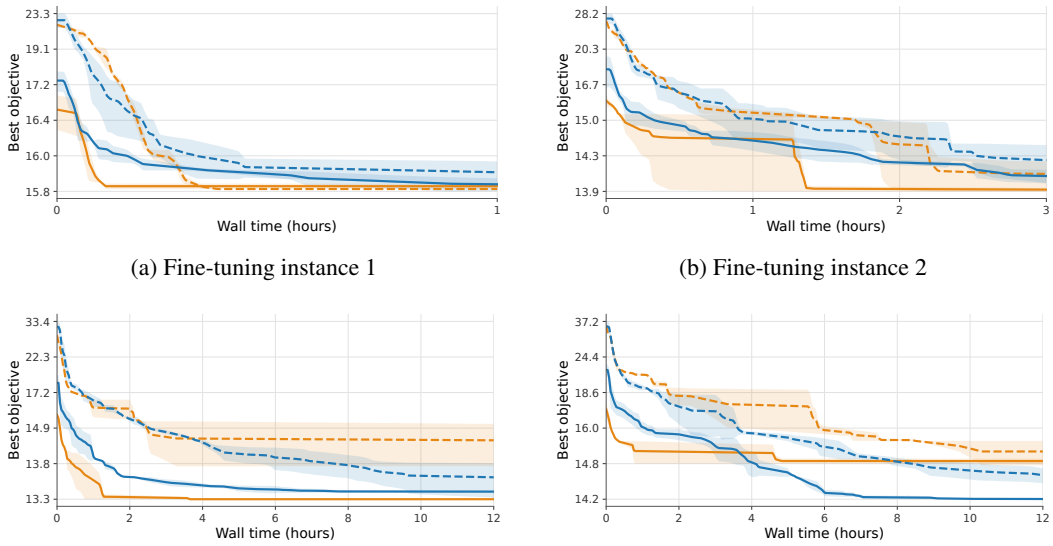
Baselines. We compare Gumbel MCTS (Danihelka et al., 2022) against classical, model-free search, and direct policy optimization baselines. *Heuristic* is a deterministic obstacle-aware, Steiner-inspired constructive method that greedily selects switch locations from the extended Hanan grid and sequentially routes communication pairs using their marginal contribution to the objective. *Random Search* (Karnopp, 1963) uniformly samples legal actions within our hierarchical framework, isolating the benefit of learned guidance. *Genetic Algorithm* (Holland, 1992) evolves complete hierarchical action sequences using selection, crossover, mutation, and random immigration, providing a stronger non-learning search baseline. Finally, *PPO-EWMA* (Hilton et al., 2022) uses the same policy-value architecture and hierarchical environment as Gumbel MCTS, but acts directly from the learned policy without tree search, isolating the contribution of explicit search.

All methods except *Heuristic* are run for 48 hours on the 24 pretraining floorplans. *Heuristic* is instead run once as a deterministic constructive procedure. Fine-tuning is performed for 24 hours, with transfer results aggregated over three independent runs. For both PPO-EWMA and Gumbel MCTS, we compare fine-tuning from the respective pretrained checkpoint against training from scratch on each of the four held-out floorplans.

6.2 Results and Discussion

Comparison with optimization baselines. Table 1 reports the objective obtained on the 24 training floorplans. The Heuristic achieves strong results on several instances, but its performance is highly instance-dependent. Random Search performs poorly throughout. This shows the need for effective guidance within the hierarchical search space. The Genetic Algorithm is considerably stronger and competitive on several instances, but struggles on more challenging floorplans, such as 18 and 19. PPO-EWMA improves upon these baselines on most instances, and shows the benefit of learned guidance, but still exhibits substantial performance gaps on some instances. Gumbel MCTS achieves the strongest and most consistent performance, including on instances where the other methods struggle. This is particularly evident on floorplans 18 and 19, where it obtains objectives of 13.601 and 14.373, compared with 19.147 and 18.488 for PPO-EWMA and 19.881 and 22.062 for the Genetic Algorithm. Among all methods, Gumbel MCTS benefits the most from additional compute, continuing to improve as the search budget increases.

Transfer to unseen floorplans. We next investigate whether training across multiple floorplans produces reusable policies that facilitate optimization of previously unseen instances. We pretrain on the 24 training floorplans and evaluate transfer on four held-out floorplans. For each target instance, we compare fine-tuning from the fixed pretrained checkpoint against training the same model from random initialization. Results are aggregated over three independent fine-tuning runs. Figure 5



(a) Fine-tuning instance 1

(b) Fine-tuning instance 2

(c) Fine-tuning instance 3

(d) Fine-tuning instance 4

Figure 5: Comparison between optimization from scratch and fine-tuning from a policy pretrained on the 24 training floorplans, evaluated on held-out floorplans. Each curve shows the mean over three independent runs, and the shaded region indicates one standard deviation. The objective is shown on a logarithmic scale. For readability, the time axis is truncated once all methods are within 1% of their respective best objective values. PPO-EWMA is shown in orange and Gumbel MCTS in blue. Solid lines correspond to fine-tuning from the pretrained policy, while dotted lines correspond to optimization from scratch.

shows that pretraining substantially accelerates optimization on held-out floorplans. Fine-tuning starts from stronger solutions than training from scratch and reaches competitive solutions using less target-instance optimization. On some instances, fine-tuning also reaches better solutions within the available optimization budget.

6.3 Limitations

Our formulation deliberately omits several aspects of physical design. The current environment does not model component dimensions, pin-level constraints, unit density, routing congestion, or bandwidth constraints, and the optimization objective considers only physical wirelength and communication-route length. Similarly, our experiments are limited to relatively small routing instances compared with real-world NoC designs, which can contain up to tens of thousands of communication connections. While the formulation does not assume fixed instance sizes, larger instances increase the computational cost. Extending the formulation to richer physical constraints and objectives, and evaluating it at industrial scale, are left to future work.

7 Conclusion

We introduced a learning-based framework for joint routing and switch placement based on a validity-preserving graph hierarchical formulation. The formulation restricts switch locations to an extended Hanan grid and enforces feasibility at every complete expansion-refinement cycle. Together, these choices discretize the placement space and restrict search to feasible solutions, substantially reducing the difficulty of the search. Experiments across diverse floorplans show that Gumbel MCTS effectively explores the resulting search space, consistently outperforming classical optimization baselines and direct policy optimization, with particularly large improvements on more challenging instances. Pretraining across multiple floorplans also provides a useful initialization for unseen instances, reducing the time required to find high-quality solutions.

Reproducibility Statement

All proofs are given in Appendix A. Appendices B and C provide all the details for the implementation and experiments. The code will be made public upon acceptance of the paper.

AI Use Statement

Generative AI was used to assist with literature review, polishing the writing, and coding. It was also used to double-check and refine the proofs in Appendix A. In particular, it provided a counterexample to an initial version of Proposition 4.8, which prompted us to refine the proposition statement. We take responsibility for the final content of this work, including all text, claims, code, and other materials produced with the assistance of generative AI.

References

Tapani Ahonen, David A Sigüenza-Tortosa, Hong Bin, and Jari Nurmi. Topology optimization for application-specific networks-on-chip. In *International Workshop on System Level Interconnect Prediction*, 2004.

Guillaume MJ-B Chaslot, Mark HM Winands, and H Jaap van Den Herik. Parallel monte-carlo tree search. In *International Conference on Computers and Games*, 2008.

Po-Yan Chen, Bing-Ting Ke, Tai-Cheng Lee, I-Ching Tsai, Tai-Wei Kung, Li-Yi Lin, En-Cheng Liu, Yun-Chih Chang, Yih-Lang Li, and Mango C-T Chao. A reinforcement learning agent for obstacle-avoiding rectilinear steiner tree construction. In *International Symposium on Physical Design*, 2022.

Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

Ruoyu Cheng and Junchi Yan. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 2021.

Ruoyu Cheng, Xianglong Lyu, Yang Li, Junjie Ye, Jianye Hao, and Junchi Yan. The policy-gradient placement and generative routing neural networks for chip design. *Advances in Neural Information Processing Systems*, 2022.

Chris Chu and Yiu-Chung Wong. Fast and accurate rectilinear steiner minimal tree algorithm for vlsi design. In *International Symposium on Physical Design*, 2005.

Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with gumbel. In *International Conference on Learning Representations*, 2022.

W.A. Dees and R.J. Smith. Performance of interconnection rip-up and reroute strategies. In *Design Automation Conference*, 1981.

Xingbo Du, Chonghua Wang, Ruizhe Zhong, and Junchi Yan. Hubrouter: Learning global routing via hub generation and pin-hub connection. *Advances in Neural Information Processing Systems*, 2023.

Xingbo Du, Ruizhe Zhong, and Junchi Yan. Train on pins and test on obstacles for rectilinear steiner minimum tree. *Advances in Neural Information Processing Systems*, 2026.

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 2022.

Michael R. Garey and David S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 1977.

Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Mingxuan Yuan, Jianye Hao, Yongdong Zhang, and Feng Wu. Reinforcement learning within tree search for fast macro placement. In *Forty-first International Conference on Machine Learning*, 2024.

Maurice Hanan. On steiner's problem with rectilinear distance. *SIAM Journal on Applied mathematics*, 1966.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *AAAI Conference on Artificial Intelligence*, 2019.

Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. *Advances in Neural Information Processing Systems*, 2022.

John H Holland. Genetic algorithms. *Scientific american*, 1992.

Jiang Hu and Sachin S. Sapatnekar. A survey on multi-net global routing for integrated circuits. *Integration*, 2001.

Edmund Ihler, Gabriele Reich, and Peter Widmayer. Class steiner trees and vlsi-design. *Discrete Applied Mathematics*, 1999.

Andrew B. Kahng, Lutong Wang, and Bangqi Xu. Tritonroute: The open-source detailed router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

Dean C Karnopp. Random search techniques for optimization problems. *Automatica*, 1963.

C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, 1961.

Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. A deep reinforcement learning approach for global routing. *Journal of Mechanical Design*, 2020.

Ralph Linsker. An iterative-improvement penalty-function-driven wire routing system. *IBM Journal of Research and Development*, 1984.

Jinwei Liu, Gengjie Chen, and Evangeline FY Young. Rest: Constructing rectilinear steiner minimum tree via reinforcement learning. In *ACM/IEEE Design Automation Conference*, 2021.

Ruizhi Liu, Zhisheng Zeng, Shizhe Ding, Jingyan Sui, Xingquan Li, and Dongbo Bu. Neuralsteiner: Learning steiner tree for overflow-avoiding global routing in chip design. *Advances in Neural Information Processing Systems*, 2024.

Radu Marculescu, Umit Y Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 2008.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, et al. A graph placement methodology for fast chip design. *Nature*, 2021.

Ahmed A Morgan, Haytham Elmiligi, Mohamed Watheq El-Kharashi, and Fayez Gebali. Unified multi-objective mapping and architecture customisation of networks-on-chip. *IET Computers & Digital Techniques*, 2013.

Yunqi Shi, Ke Xue, Song Lei, and Chao Qian. Macro placement by wire-mask-guided black-box optimization. *Advances in Neural Information Processing Systems*, 2023.

Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. A survey on steiner tree construction and global routing for vlsi design. *IEEE Access*, 2020.

Appendix

Table of Contents

A Proofs **13**
A.1 Optimality on the Hanan Grid 13
A.2 Correct and Static Routing at Every Step of the Algorithm 14
A.3 Optimality of the Hierarchical Search Space 14

B Implementation Details **16**
B.1 Model Architecture 16
B.2 PPO-EWMA 16
B.3 Gumbel MCTS 17

C Experimental Details **17**
C.1 Datasets and Transfer Protocol 17
C.2 Optimization Baselines 18
C.3 Compute and Evaluation 18

D Detailed Numerical Results **19**
D.1 Pretraining 19
D.2 Fine-Tuning 21

E Floorplans Generated by the Different Methods **22**
E.1 Pretraining 22
E.2 Fine-Tuning 46

A PROOFS

In this section, we provide proofs for all the propositions in the main paper.

A.1 OPTIMALITY ON THE HANAN GRID

Proposition A.1 (Optimal Switch Placement)**.** *Under rectilinear routing, and for fixed positions of initiators, targets, and blockages, there exists an optimal solution in which every switch is placed at an intersection of the extended Hanan grid $\mathcal{H}$.*

Proof. We first establish a geometric property used in the argument. Recall that blockage interiors are forbidden, while their boundaries are admissible.

Lemma A.2. *Let C be the set of switches sharing an off-grid x-coordinate x, and let $x^- < x < x^+$ be consecutive coordinates obtained by augmenting the vertical Hanan lines with all switch x-coordinates. Every physical connection incident to C admits a shortest realization whose length varies affinely when C is translated horizontally within $[x^-, x^+]$.*

Proof. Since x^- and x^+ are consecutive augmented coordinates, no vertical blockage boundary lies in the open slab $(x^-, x^+) \times \mathbb{R}$. Consequently, from any feasible point (x, y) in the slab, the horizontal segment to either boundary is feasible: otherwise a rectangular blockage would have a vertical boundary inside the slab or would contain (x, y).

Consider first a shortest connection from $s = (x, y_s) \in C$ to an endpoint outside C. The other endpoint cannot lie strictly inside the slab, since terminal coordinates are Hanan coordinates and all switch coordinates were included in the augmentation. Hence the connection reaches one of the slab boundaries; suppose it first reaches x^+ at $b = (x^+, y_b)$.

The prefix from s to b can be replaced by a horizontal segment from (x, y_s) to (x^+, y_s) followed by a vertical segment along x^+ to b. This path is feasible and has length $x^+ - x + |y_b - y_s|$, the rectilinear lower bound between s and b. It is therefore also shortest. After translating s by δ, only its initial horizontal segment changes, so the connection length changes by $-\delta$. A connection reaching x^- analogously changes by $+\delta$.

Now consider a connection whose two endpoints belong to C. If their vertical segment is feasible, it remains feasible under a common translation within the slab and its length is constant. Otherwise, any feasible connection between them must leave the slab. Applying the preceding argument at both endpoints gives a shortest realization with fixed boundary attachments. Its length therefore changes by -2δ, 2δ, or 0, according to the boundaries through which it leaves and re-enters the slab.

Thus every connection incident to C has a shortest realization whose length is affine in δ. □

We now consider a globally optimal solution with objective $J^\star$ and fix its logical routing topology. For each physical connection $\{u, v\}$, let m_{uv} be the number of communication routes traversing it. Its contribution to the objective has fixed positive weight $w_{uv} = 1 + \lambda m_{uv}$. Hence, for the fixed topology, the optimal objective can be written as $J^\star = \sum_{\{u,v\} \in E} w_{uv}\, d_{\mathcal{B}}(u, v)$, where $d_{\mathcal{B}}(u, v)$ denotes the shortest obstacle-avoiding rectilinear distance between u and v.

We first align the x-coordinates. Let $\mathcal{X}$ be the set of vertical Hanan coordinates and define $\widehat{\mathcal{X}} = \mathcal{X} \cup \{x_s : s \in \mathcal{S}\}$. Group switches sharing the same $x \notin \mathcal{X}$, and let N_x be the number of such off-grid groups.

Consider one group C at x, with adjacent coordinates $x^- < x < x^+$ in $\widehat{\mathcal{X}}$. Choose for each incident connection a shortest realization given by Lemma A.2. If C is translated by δ, with $x + \delta \in [x^-, x^+]$, the weighted length of these realizations is affine: $\widetilde{J}(\delta) = J(0) + a\delta$ for some constant a. Since $\delta = 0$ lies between the two boundary displacements, at least one boundary satisfies $\widetilde{J}(\delta) \le J(0)$.

Move C to such a boundary. The constructed paths certify a feasible solution with no larger objective; replacing them by shortest paths at the new switch positions can only improve it further. If the reached coordinate is in $\mathcal{X}$, C is now Hanan-aligned; otherwise it merges with another switch group.

In either case, N_x strictly decreases. Repeating this operation therefore places every switch on a vertical Hanan line after finitely many moves, without increasing the objective.

Applying the same argument to the y-coordinates, while keeping the x-coordinates fixed, places every switch at an intersection of $\mathcal{H}$ without increasing the objective. The paths maintained during these translations are feasible, but need not be shortest at the final switch positions. We therefore replace each physical connection by a shortest obstacle-avoiding rectilinear path between its final endpoints. Such a path exists because the maintained path provides a feasible connection, and this replacement can only decrease the objective. Thus, starting from the globally optimal value $J^\star$, we obtain a feasible grid-aligned solution with objective at most $J^\star$. By global optimality, its objective must equal $J^\star$, completing the proof. □

A.2 Correct and Static Routing at Every Step of the Algorithm

Proposition A.3 (Validity preservation)**.** *Assume the routing configuration before expansion assigns exactly one simple directed path to every communication pair. After applying switch expansion and route refinement (Definitions 4.3 and 4.5), the resulting routing configuration again assigns exactly one simple directed path to every communication pair.*

Proof. Let $G = (\mathcal{V}, \mathcal{E})$ be a valid routing configuration and let s_1 be the switch selected for expansion. Consider a communication pair $(i, t) \in \mathcal{R}$.

If $\pi_{i,t}$ does not traverse s_1, neither switch expansion nor route refinement modifies its route, so $\pi_{i,t}$ remains a unique simple directed path.

Now suppose that $\pi_{i,t}$ traverses s_1. Because $\pi_{i,t}$ is simple, it visits s_1 exactly once. Let r_b and r_a denote the route nodes immediately preceding and following s_1 on $\pi_{i,t}$. Switch expansion leaves the remainder of $\pi_{i,t}$ unchanged and replaces the local segment $r_b \to s_1 \to r_a$ by the four alternatives $\rho_1 : r_b \to s_1 \to r_a$, $\rho_2 : r_b \to s_2 \to r_a$, $\rho_3 : r_b \to s_1 \to r_m \to s_2 \to r_a$ and $\rho_4 : r_b \to s_2 \to r_m \to s_1 \to r_a$, where s_2 is the newly introduced switch and r_m is the corresponding route node for (i, t).

Each ρ_j is a simple directed path from r_b to r_a. Indeed, s_2 and r_m are newly introduced nodes, while s_1 occurs only once in each alternative. Route refinement selects exactly one ρ_j and removes the edges belonging exclusively to the remaining alternatives. Replacing the original local segment by the selected ρ_j therefore yields exactly one simple directed path from i to t.

Since this argument applies independently to every communication pair affected by the expansion, while all unaffected routes remain unchanged, the refined configuration assigns exactly one simple directed path to every communication pair. □

A.3 Optimality of the Hierarchical Search Space

Proposition A.4 (Optimality of the hierarchical search space)**.** *There exists a globally optimal solution to Equation* (1) *that can be obtained from the initial configuration through a finite sequence of switch expansions, placements, and route refinements.*

Proof. We first establish a structural property of optimal routing solutions. For a communication route π containing two switches s_a and s_b, let $\pi[s_a, s_b]$ denote the physical subpath between them, independently of its traversal direction.

Lemma A.5 (Common-subpath property)**.** *There exists a globally optimal routing configuration such that, for any pair of switches s_a and s_b, all communication routes containing both switches use the same switch sequence between them, possibly in reverse order.*

Proof. Index the switches up to the budget $S_{\max}$, and assign every possible undirected connection e between terminals and switches a positive tie-breaking weight $\omega_e = 2^{k(e)}$, where the exponents $k(e)$ are distinct. For a route $\pi = (v_0, \ldots, v_k)$, define $\tau(\pi) = \sum_{j=1}^{k} \omega_{\{v_{j-1}, v_j\}}$. Since routes are simple, two distinct paths between the same endpoints, up to reversal, have different values of τ.

Among all globally optimal routing configurations, choose one minimizing $\Psi = \sum_{(i,t)\in\mathcal{R}} \tau(\pi_{i,t})$. Such a configuration exists because the switch budget and the number of communication pairs are finite, and hence only finitely many simple route sequences are possible.

Suppose, for contradiction, that two communication routes contain the same switches s_a and s_b but use different switch sequences P_1 and P_2 between them. Orient both sequences from s_a to s_b, and let $L(P)$ denote their route-length contribution. Without loss of generality, assume that $L(P_1) < L(P_2)$, or that $L(P_1) = L(P_2)$ and $\tau(P_1) < \tau(P_2)$.

Replace P_2 in its route by P_1, using the reverse of P_1 if necessary. Every connection of P_1 is already present in the routing configuration, so this introduces no new physical wire. If the resulting route contains a repeated switch, erase the resulting loops; this preserves connectivity and can only remove physical wire and route length.

If $L(P_1) < L(P_2)$, the route-length term strictly decreases while wirelength does not increase, contradicting global optimality. Otherwise, the objective does not increase and therefore, by global optimality, remains unchanged. The resulting configuration is thus also globally optimal, but replacing P_2 by P_1 strictly decreases Ψ. Any loop removal only decreases it further because all ω_e are positive. This contradicts the choice of the globally optimal configuration minimizing Ψ.

Therefore no such pair of routes can exist, proving the claim. □

Now let $G^\star$ be such a globally optimal configuration. By Proposition 4.2, we may additionally choose $G^\star$ such that every switch is placed on the extended Hanan grid $\mathcal{H}$.

We now construct a finite sequence of contractions $G^\star = G^{(m)} \xrightarrow{C_m} G^{(m-1)} \xrightarrow{C_{m-1}} \cdots \xrightarrow{C_1} G^{(0)}$, where $G^{(0)}$ contains a single switch.

Consider a configuration $G^{(k)}$ containing at least two switches. If some communication route contains at least two switches, choose two switches s_a and s_b that are consecutive on that route. Their physical subpath is the single connection between s_a and s_b; hence, by Lemma A.5, every other communication route containing both switches also contains them consecutively, possibly in the opposite order. If no communication route contains two switches, choose any two remaining switches s_a and s_b, which cannot occur together on any communication route.

For every communication pair whose route contains at least one of s_a and s_b, the local subpath involving these switches is therefore one of $\to s_a \to$, $\to s_b \to$, $\to s_a \to s_b \to$, $\to s_b \to s_a \to$. Before contracting the switches, we record this local subpath for each such communication pair.

The contraction C_k identifies s_a and s_b with a single macro-switch s and replaces each of these local subpaths by $\to s \to$. All other communication routes are left unchanged. Each contraction removes one switch. Repeating this construction therefore yields, after finitely many steps, a single-switch configuration $G^{(0)}$.

We now reverse the sequence constructively. For each contraction C_k, we recorded the two switches (s_a, s_b), their positions in $G^\star$, and the local subpath of every communication route affected by the contraction. Starting from $G^{(k-1)}$, we expand the corresponding macro-switch s into s_a and s_b and assign them their positions $\mathbf{p}^\star_{s_a}, \mathbf{p}^\star_{s_b} \in \mathcal{H}$. For each communication pair traversing s, route refinement replaces $\to s \to$ by its recorded local subpath, $\to s_a \to$, $\to s_b \to$, $\to s_a \to s_b \to$, or $\to s_b \to s_a \to$. All other communication routes are unaffected. Thus, the expansion, placement, and refinement operations reconstruct $G^{(k)}$ exactly from $G^{(k-1)}$.

Reversing all contractions consequently gives $G^{(0)} \longrightarrow G^{(1)} \longrightarrow \cdots \longrightarrow G^{(m)} = G^\star$, where every transition consists of one switch expansion, the corresponding switch placements, and the recorded route refinements. Hence $G^\star$ can be constructed from the initial configuration through a finite sequence of switch expansions, placements, and route refinements. Since $G^\star$ is globally optimal, the hierarchical search space contains a globally optimal solution. □

B Implementation Details

B.1 Model Architecture

The policy and value functions share a neural architecture that jointly encodes the current routing configuration and physical floorplan. The routing configuration is represented as an augmented directed graph containing initiators, targets, instantiated switches, and route-bundle nodes representing physical segments shared by one or more communication routes. The floorplan is represented by a raster encoding of the blockage geometry and legal candidate switch locations.

The architecture is structured as follows:

1. **Routing graph representation:** Graph nodes represent initiators, targets, switches, and route bundles. Node and edge attributes encode node type, normalized coordinates, communication identities, active switches, geometric distances, and the current routing configuration.
2. **Graph feature embedding:** Coordinate and categorical attributes are independently embedded into 24-dimensional representations and combined to initialize the graph features used by the policy and value networks.
3. **Floorplan encoding:** The physical floorplan is represented by a two-channel 128×128 image. The first channel encodes blockage occupancy, while the second identifies legal candidate switch locations. A convolutional encoder consisting of four blocks with GroupNorm and ReLU activations produces spatial and global floorplan representations. These representations condition both the routing-graph features and the switch-placement predictions.
4. **Graph processing:** The routing graph is processed by four directed message-passing layers with hidden dimension 80. This produces contextual node representations that combine the current routing structure with the encoded physical environment.
5. **Policy prediction:** Separate policy heads parameterize the three action types of the hierarchical MDP: switch expansion, switch placement, and route refinement. For expansion, the policy predicts a scalar score for each existing switch, normalized across the switches in the current graph. For placement, it predicts a two-dimensional score map over candidate locations for each switch being placed. Invalid and absent locations are masked, and each placement map is normalized independently. For refinement, the policy predicts four scores for each relevant graph edge, corresponding to the four refinement alternatives defined in Sec. 4. Predictions associated with edges belonging to the same affected communication route are averaged to obtain its four refinement scores.
6. **Value prediction:** The contextual node representations are aggregated using attention pooling with four learned queries and passed through a value network with two hidden layers of dimension 160. For PPO-EWMA, the network predicts a scalar state value. For Gumbel MCTS, following AlphaTensor (Fawzi et al., 2022), it predicts eight return quantiles, with the mean of the two largest quantiles used as the leaf value during search. The quantile outputs are trained using the quantile Huber loss at levels $\tau_i = (i - \frac{1}{2})/8$.

   We use PopArt normalization with separate statistics for each floorplan. Statistics are updated once per fresh trajectory collection using a decay of 0.9 and a minimum standard deviation of 10^{-6}. Following each update, the output-layer parameters are rescaled so that the corresponding unnormalized value predictions remain unchanged.

The complete policy and value architecture contains approximately 0.7 million trainable parameters. Both learning methods use AdamW with learning rate 10^{-4} and zero weight decay, per-GPU optimization batches of 4096, and gradient-norm clipping at 1.

B.2 PPO-EWMA

We train PPO using an exponentially weighted moving-average proximal policy (Hilton et al., 2022). Complete trajectories are collected under the current behavior policy and optimized using a clipped importance-sampling objective relative to the exponentially averaged proximal policy. We use undiscounted Monte Carlo returns with $\gamma = 1$.

We use one optimization epoch per rollout, a value-loss coefficient of 0.5, an entropy coefficient of 0.05, a clipping coefficient of 0.01, a proximal-policy EWMA decay of 0.889, and a maximum

importance ratio of 100. Advantages are normalized separately for each floorplan using exponentially weighted running moments with decay 0.9. We collect 8192 trajectories per GPU per collection.

B.3 GUMBEL MCTS

We implement Gumbel MCTS following Gumbel AlphaZero (Danihelka et al., 2022). At each decision state, the policy network provides action priors and the value network predicts eight return quantiles. The mean of the largest two predicted quantiles, corresponding to the upper quartile of the predicted return distribution, is used as the scalar leaf value during search. At the root, actions are selected using Gumbel-perturbed policy logits and evaluated through sequential halving. At non-root states, simulations are allocated according to the completed-value policy-improvement rule.

During training, we use 800 MCTS simulations per decision and collect 512 trajectories per floorplan per collection. At most 128 root actions are retained for sequential halving. We use unit-scale Gumbel perturbations, $c_{\text{visit}} = 50$, and $c_{\text{scale}} = 0.01$.

Search simulations are evaluated asynchronously. Within each root-action subtree, the number of concurrent simulations is limited to 0.08 of the simulation budget. A virtual loss (Chaslot et al., 2008) of 0.1, expressed in normalized completed-Q units, is applied to in-flight branches to reduce collisions between concurrent simulations.

To stabilize learning from search-improved targets, we interpolate the policy target with the current policy using $\eta_\pi = 0.25$: $\pi_{\text{target}} = (1 - \eta_\pi)\pi_{\text{prior}} + \eta_\pi \pi_{\text{search}}$.

For the quantile critic, let $\mathbf{z}_{\text{prior}} = (z_1, \ldots, z_8)$ denote the predicted return quantiles and let G denote the realized return. At collection k, each quantile target is

$$z_{i,\text{target}} = (1 - \eta_V^{(k)}) z_{i,\text{prior}} + \eta_V^{(k)} G, \qquad \eta_V^{(k)} = \max\left(0.25, \frac{1}{k}\right). \tag{3}$$

The larger interpolation coefficient during the initial collections mitigates critic cold start; afterward, it matches the policy interpolation coefficient.

MCTS samples are retained for two collections, corresponding to an expected 20 optimization replays per sample. The value-loss coefficient is 0.5.

C EXPERIMENTAL DETAILS

This section provides additional details on the datasets, baselines, training setup, computational resources, and evaluation protocols used in our experiments.

Unless otherwise stated, all methods are evaluated using the same routing objective and feasibility requirements. Random Search and the Genetic Algorithm operate directly in the same hierarchical routing environment as the learning-based methods. The Heuristic uses the same extended Hanan grid. PPO-EWMA and Gumbel MCTS share the same routing environment and neural backbone, with scalar and quantile value heads, respectively.

C.1 DATASETS AND TRANSFER PROTOCOL

Our pretraining dataset contains 24 synthetically generated routing floorplans. Each instance contains between 6 and 18 rectangular blockages, 3–5 initiators, 5–8 targets, and 18–25 required directed communications. The switch budget ranges from 2 to 5. All floorplans are square.

For each learning algorithm, a single shared model is trained across all pretraining floorplans, with trajectories distributed approximately uniformly across instances during data collection.

For transfer experiments, we use four target floorplans held out from pretraining, covering two-, three-, and four-switch routing settings. For each target instance, we compare initialization from the corresponding pretrained model against training from scratch. When transferring a pretrained model, we retain all shape-compatible shared parameters, including the floorplan encoder, message-passing layers, policy heads, attention pooling, and hidden value-network layers. The initiator-, target-, and route-identifier embeddings and the floorplan-specific PopArt output layer are reinitialized. Optimizer state, replay data, PopArt statistics, and algorithm-specific training state are not transferred.

C.2 Optimization Baselines

Heuristic. The Heuristic is a deterministic obstacle-aware, Steiner-inspired constructive method operating on the extended Hanan grid. Every communication route is required to traverse at least one switch. Candidate locations are ranked by evaluating single-switch routing solutions, and the best 128 candidates are retained. The method evaluates the retained one-switch solutions and greedily expands the best partial solution by adding switches until the prescribed maximum budget is reached. The best solution encountered across all intermediate switch counts is retained, so the returned solution may use fewer switches than the maximum budget.

For each candidate switch set, communication pairs are inserted sequentially using obstacle-aware shortest paths that minimize their marginal contribution to the objective. For a segment of length d, introducing a new physical connection incurs $d + 0.5d$, whereas reusing an existing physical connection incurs only the additional route cost $0.5d$. We evaluate eight deterministic demand orderings and retain the best resulting network. Finally, one local-refinement pass considers single-switch replacements and accepts strictly improving configurations.

Random Search. Random Search operates directly on the same hierarchical construction process as the learning-based methods, but uses no learned policy or value function. At each state, it samples uniformly from the currently admissible actions, thereby producing feasible routing configurations by construction. It evaluates $18{,}000$ complete trajectories per GPU and generation and retains the best solution found.

Genetic Algorithm. The Genetic Algorithm represents each candidate solution as a sequence of hierarchical decisions and evaluates it using the same routing environment. Starting from a random population, subsequent generations combine elitist selection, tournament selection, one-point crossover, per-decision mutation, and random immigration. Because action legality depends on preceding decisions, inherited actions that are no longer admissible are replaced by uniformly sampled legal actions. We use an elite pool of 64 programs per floorplan, tournament size 4, crossover probability 0.9, mutation probability 0.05, and a 10% random immigrant fraction. Each generation evaluates $18{,}000$ complete trajectories per GPU.

C.3 Compute and Evaluation

Pretraining is performed on six NVIDIA L40S GPUs. PPO-EWMA, Gumbel MCTS, Random Search, and the Genetic Algorithm are each run for 48 hours on the 24 pretraining floorplans, with computation distributed approximately uniformly across instances. The deterministic Heuristic is run once for each floorplan. Due to the computational cost of multi-floorplan pretraining, we train one pretrained model for each learning algorithm.

Fine-tuning is performed for 24 hours on a single NVIDIA L40S GPU. Transfer results are aggregated over three independent fine-tuning runs for each target floorplan and initialization setting. All pretrained fine-tuning runs for a given learning algorithm are initialized from the same fixed pretrained checkpoint.

D Detailed Numerical Results

This section provides detailed numerical results for all experiments. For each floorplan, we additionally report its main characteristics, including the switch budget, number of initiators, targets, and communication routes, and free space, defined as the percentage of the floorplan area not covered by blockages. We report the route length, wirelength, and best objective found by each method within its corresponding optimization budget. We similarly report transfer results on the four held-out floorplans.

D.1 Pretraining

Table 2: Detailed numerical results for Heuristic on the 24 pretraining floorplans.

Heuristic								
Instance	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space (%)**	**Route length ↓**	**Wirelength ↓**	**Objective ↓**
1	4	5	8	21	71.9	16.842	3.481	11.902
2	4	4	7	18	68.9	15.975	3.267	11.255
3	4	5	8	23	71.0	19.626	5.536	15.349
4	4	5	8	22	77.2	15.623	3.518	11.330
5	4	5	8	19	76.1	17.938	5.878	14.847
6	4	5	8	22	81.0	15.640	3.306	11.126
7	4	5	8	22	75.0	9.796	3.192	8.090
8	4	5	8	23	77.7	14.270	3.166	10.301
9	4	5	8	21	78.5	15.834	3.375	11.292
10	4	5	8	23	81.3	20.724	4.253	14.615
11	3	4	5	19	68.0	21.356	4.063	14.741
12	3	4	5	19	74.9	19.355	3.740	13.418
13	3	4	5	19	75.2	19.670	3.827	13.662
14	3	4	5	19	72.2	20.028	4.989	15.003
15	2	3	7	20	68.5	19.366	4.829	14.512
16	2	3	7	20	68.7	21.211	4.679	15.284
17	2	3	7	20	63.7	23.431	4.855	16.571
18	5	5	5	25	66.5	25.126	3.803	16.366
19	4	5	5	25	76.2	24.959	3.797	16.277
20	2	4	5	19	60.6	22.797	4.879	16.277
21	4	4	5	19	58.4	20.369	4.724	14.909
22	3	4	5	19	63.2	20.309	5.260	15.415
23	4	4	5	19	61.8	18.653	4.681	14.007
24	2	3	7	20	63.1	20.961	5.203	15.684

Table 3: Detailed numerical results for Random search on the 24 pretraining floorplans.

Random search								
Instance	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space (%)**	**Route length ↓**	**Wirelength ↓**	**Objective ↓**
1	4	5	8	21	71.9	18.540	9.436	18.706
2	4	4	7	18	68.9	23.403	9.098	20.799
3	4	5	8	23	71.0	24.834	11.625	24.042
4	4	5	8	22	77.2	18.655	8.092	17.420
5	4	5	8	19	76.1	23.366	11.946	23.629
6	4	5	8	22	81.0	20.092	9.264	19.310
7	4	5	8	22	75.0	13.404	5.970	12.672
8	4	5	8	23	77.7	18.610	9.269	18.574
9	4	5	8	21	78.5	19.788	8.769	18.663
10	4	5	8	23	81.3	26.542	11.660	24.931
11	3	4	5	19	68.0	25.169	9.999	22.584
12	3	4	5	19	74.9	24.155	8.698	20.775
13	3	4	5	19	75.2	24.537	9.157	21.425
14	3	4	5	19	72.2	24.847	9.451	21.875
15	2	3	7	20	68.5	22.518	8.079	19.338
16	2	3	7	20	68.7	25.037	7.627	20.146
17	2	3	7	20	63.7	25.112	8.387	20.943
18	5	5	5	25	66.5	33.287	14.167	30.810
19	4	5	5	25	76.2	34.856	11.602	29.030
20	2	4	5	19	60.6	26.344	7.249	20.421
21	4	4	5	19	58.4	27.774	9.796	23.683
22	3	4	5	19	63.2	25.547	8.564	21.337
23	4	4	5	19	61.8	24.468	10.223	22.457
24	2	3	7	20	63.1	25.941	7.613	20.583

Table 4: Detailed numerical results for Genetic algorithm on the 24 pretraining floorplans.

Genetic algorithm								
Instance	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space** (%)	**Route length** ↓	**Wirelength** ↓	**Objective** ↓
1	4	5	8	21	71.9	17.970	5.397	14.382
2	4	4	7	18	68.9	16.293	5.853	13.999
3	4	5	8	23	71.0	22.228	5.740	16.854
4	4	5	8	22	77.2	15.589	4.827	12.622
5	4	5	8	19	76.1	20.256	7.842	17.970
6	4	5	8	22	81.0	14.812	5.701	13.107
7	4	5	8	22	75.0	9.982	3.233	8.224
8	4	5	8	23	77.7	15.332	4.288	11.954
9	4	5	8	21	78.5	17.572	5.366	14.152
10	4	5	8	23	81.3	19.600	5.999	15.799
11	3	4	5	19	68.0	21.500	5.728	16.478
12	3	4	5	19	74.9	21.160	5.924	16.504
13	3	4	5	19	75.2	18.885	5.796	15.238
14	3	4	5	19	72.2	20.606	4.754	15.057
15	2	3	7	20	68.5	19.638	4.315	14.134
16	2	3	7	20	68.7	21.019	4.727	15.236
17	2	3	7	20	63.7	20.303	5.087	15.239
18	5	5	5	25	66.5	25.411	7.175	19.881
19	4	5	5	25	76.2	29.455	7.334	22.062
20	2	4	5	19	60.6	21.177	5.138	15.726
21	4	4	5	19	58.4	19.118	7.021	16.580
22	3	4	5	19	63.2	19.422	4.821	14.531
23	4	4	5	19	61.8	19.822	4.710	14.621
24	2	3	7	20	63.1	21.269	4.643	15.277

Table 5: Detailed numerical results for PPO-EWMA on the 24 pretraining floorplans.

PPO-EWMA								
Instance	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space** (%)	**Route length** ↓	**Wirelength** ↓	**Objective** ↓
1	4	5	8	21	71.9	17.150	3.591	12.166
2	4	4	7	18	68.9	15.969	3.499	11.484
3	4	5	8	23	71.0	20.328	5.506	15.670
4	4	5	8	22	77.2	15.109	4.021	11.576
5	4	5	8	19	76.1	19.440	4.997	14.717
6	4	5	8	22	81.0	15.130	3.395	10.960
7	4	5	8	22	75.0	10.702	3.244	8.595
8	4	5	8	23	77.7	14.482	3.111	10.352
9	4	5	8	21	78.5	15.860	3.212	11.142
10	4	5	8	23	81.3	19.434	3.612	13.329
11	3	4	5	19	68.0	20.563	3.869	14.151
12	3	4	5	19	74.9	19.274	3.724	13.361
13	3	4	5	19	75.2	18.415	4.080	13.287
14	3	4	5	19	72.2	19.741	3.722	13.593
15	2	3	7	20	68.5	19.638	4.315	14.134
16	2	3	7	20	68.7	20.543	4.998	15.269
17	2	3	7	20	63.7	20.303	5.087	15.239
18	5	5	5	25	66.5	27.654	5.320	19.147
19	4	5	5	25	76.2	27.178	4.899	18.488
20	2	4	5	19	60.6	22.841	4.396	15.817
21	4	4	5	19	58.4	18.409	3.890	13.095
22	3	4	5	19	63.2	20.314	3.832	13.989
23	4	4	5	19	61.8	18.419	3.199	12.409
24	2	3	7	20	63.1	21.269	4.643	15.277

Table 6: Detailed numerical results for Gumbel MCTS on the 24 pretraining floorplans.

Gumbel MCTS								
Instance	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space (%)**	**Route length ↓**	**Wirelength ↓**	**Objective ↓**
1	4	5	8	21	71.9	16.782	2.942	11.333
2	4	4	7	18	68.9	15.463	3.148	10.880
3	4	5	8	23	71.0	19.616	4.466	14.274
4	4	5	8	22	77.2	13.849	2.859	9.784
5	4	5	8	19	76.1	19.014	4.419	13.926
6	4	5	8	22	81.0	14.496	2.670	9.918
7	4	5	8	22	75.0	9.788	2.845	7.739
8	4	5	8	23	77.7	14.084	2.855	9.897
9	4	5	8	21	78.5	15.784	2.928	10.820
10	4	5	8	23	81.3	19.168	3.444	13.028
11	3	4	5	19	68.0	20.850	3.570	13.995
12	3	4	5	19	74.9	19.274	3.724	13.361
13	3	4	5	19	75.2	18.415	4.080	13.287
14	3	4	5	19	72.2	19.870	3.531	13.466
15	2	3	7	20	68.5	19.638	4.315	14.134
16	2	3	7	20	68.7	21.019	4.727	15.236
17	2	3	7	20	63.7	20.303	5.087	15.239
18	5	5	5	25	66.5	22.272	2.465	13.601
19	4	5	5	25	76.2	23.025	2.861	14.373
20	2	4	5	19	60.6	21.177	5.138	15.726
21	4	4	5	19	58.4	18.801	3.379	12.779
22	3	4	5	19	63.2	19.669	3.637	13.471
23	4	4	5	19	61.8	18.393	3.190	12.386
24	2	3	7	20	63.1	21.269	4.643	15.277

D.2 FINE-TUNING

Table 7: Detailed PPO-EWMA fine-tuning results with pretrained and scratch initialization. Values are mean ± standard deviation over three runs.

Instance	**Initialization**	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space (%)**	**Route length ↓**	**Wirelength ↓**	**Objective ↓**
1	Pretrained	2	5	6	20	76.1	$21.310_{\pm 0.020}$	$5.165_{\pm 0.000}$	$15.820_{\pm 0.010}$
	Scratch						$21.290_{\pm 0.000}$	$5.165_{\pm 0.000}$	$15.810_{\pm 0.000}$
2	Pretrained	3	5	6	20	72.9	$18.765_{\pm 0.010}$	$4.550_{\pm 0.005}$	$13.933_{\pm 0.000}$
	Scratch						$18.765_{\pm 0.010}$	$4.550_{\pm 0.005}$	$13.933_{\pm 0.000}$
3	Pretrained	4	5	6	20	71.1	$18.045_{\pm 0.000}$	$4.325_{\pm 0.000}$	$13.347_{\pm 0.000}$
	Scratch						$20.115_{\pm 0.050}$	$4.222_{\pm 0.477}$	$14.280_{\pm 0.503}$
4	Pretrained	4	5	5	25	69.1	$23.554_{\pm 0.139}$	$3.048_{\pm 0.126}$	$14.825_{\pm 0.057}$
	Scratch						$24.000_{\pm 0.304}$	$3.079_{\pm 0.198}$	$15.078_{\pm 0.349}$

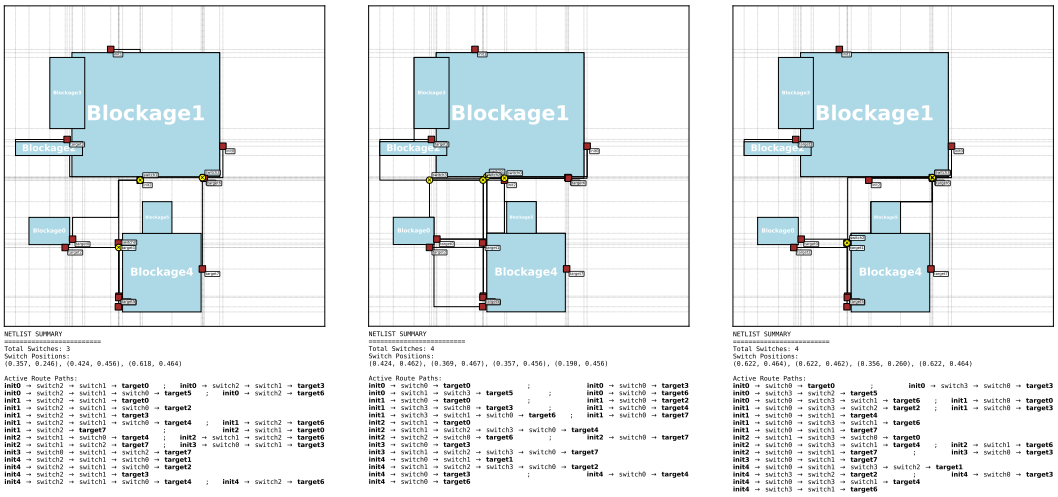
Table 8: Detailed Gumbel MCTS fine-tuning results with pretrained and scratch initialization. Values are mean ± standard deviation over three runs.

Instance	**Initialization**	**Switch budget**	**Initiators**	**Targets**	**Routes**	**Free space (%)**	**Route length ↓**	**Wirelength ↓**	**Objective ↓**
1	Pretrained	2	5	6	20	76.1	$21.290_{\pm 0.000}$	$5.165_{\pm 0.000}$	$15.810_{\pm 0.000}$
	Scratch						$21.290_{\pm 0.000}$	$5.165_{\pm 0.000}$	$15.810_{\pm 0.000}$
2	Pretrained	3	5	6	20	72.9	$20.135_{\pm 0.008}$	$3.903_{\pm 0.002}$	$13.971_{\pm 0.005}$
	Scratch						$20.365_{\pm 0.339}$	$3.970_{\pm 0.092}$	$14.153_{\pm 0.262}$
3	Pretrained	4	5	6	20	71.1	$18.895_{\pm 0.523}$	$3.970_{\pm 0.264}$	$13.417_{\pm 0.012}$
	Scratch						$19.532_{\pm 0.207}$	$3.815_{\pm 0.142}$	$13.581_{\pm 0.245}$
4	Pretrained	4	5	5	25	69.1	$20.620_{\pm 0.000}$	$3.907_{\pm 0.000}$	$14.217_{\pm 0.000}$
	Scratch						$23.134_{\pm 0.215}$	$2.852_{\pm 0.009}$	$14.419_{\pm 0.116}$

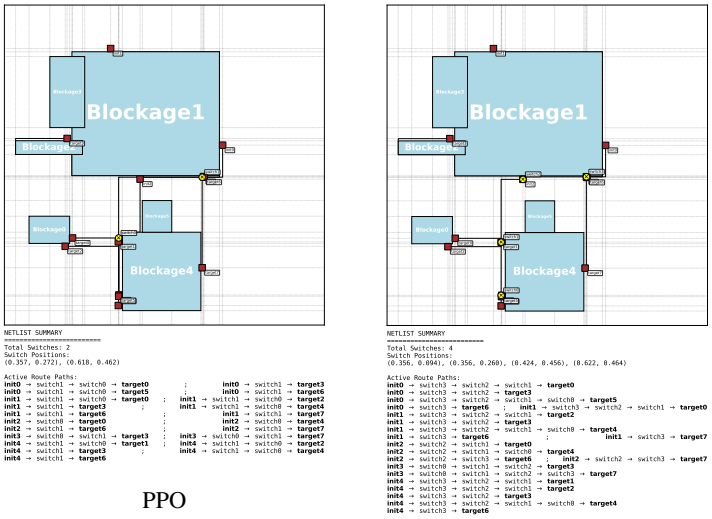
E FLOORPLANS GENERATED BY THE DIFFERENT METHODS

This section provides the solution corresponding to the best objective value found for each experiment and method.

E.1 PRETRAINING

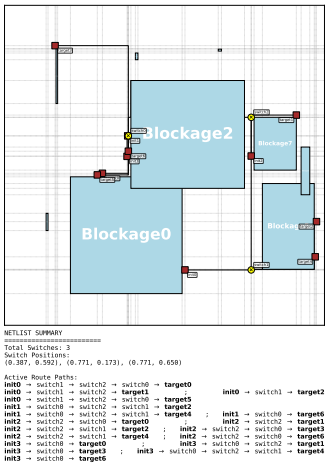


Heuristic Random search Genetic algorithm



PPO MCTS

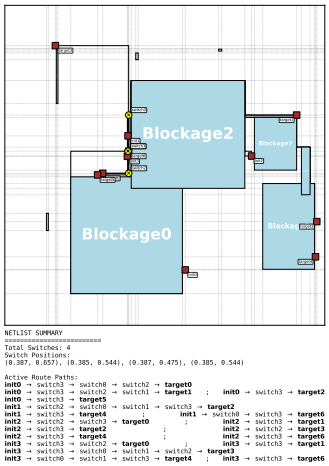
Figure 6: Instance 1.



NETLIST SUMMARY
Total Switches: 3
Switch Positions:
(0.387, 0.592), (0.771, 0.173), (0.771, 0.658)

Active Route Paths:
init0 → switch1 → switch2 → switch0 → target0
init0 → switch1 → switch2 → target1 ; init0 → switch1 → target2
init0 → switch1 → switch2 → switch0 → target5
init1 → switch0 → switch2 → switch1 → target2
init1 → switch0 → switch2 → switch1 → target4 ; init1 → switch0 → target6
init2 → switch2 → switch0 → target0 init2 → switch2 → target1
init2 → switch2 → switch1 → target2 ; init2 → switch2 → switch0 → target3
init2 → switch2 → switch1 → target4 ; init2 → switch2 → switch0 → target6
init3 → switch0 → target0 ; init3 → switch0 → switch2 → target1
init3 → switch0 → target3 ; init3 → switch0 → switch2 → switch1 → target4
init3 → switch0 → target6

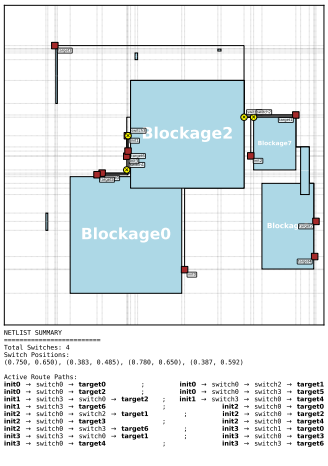
Heuristic



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.387, 0.657), (0.385, 0.544), (0.387, 0.475), (0.385, 0.544)

Active Route Paths:
init0 → switch3 → switch0 → switch2 → target0
init0 → switch3 → switch2 → switch1 → target1 ; init0 → switch3 → target2
init0 → switch3 → target5
init1 → switch2 → switch0 → switch1 → switch3 → target2
init1 → switch3 → target4 ; init1 → switch0 → switch3 → target6
init2 → switch2 → switch3 → target0 init2 → switch3 → target1
init2 → switch3 → target2 ; init2 → switch2 → target3
init2 → switch3 → target4 ; init2 → switch3 → target6
init3 → switch3 → switch2 → target0 ; init3 → switch3 → target1
init3 → switch3 → switch0 → switch1 → switch2 → target3
init3 → switch0 → switch1 → switch3 → target4 ; init3 → switch3 → target6

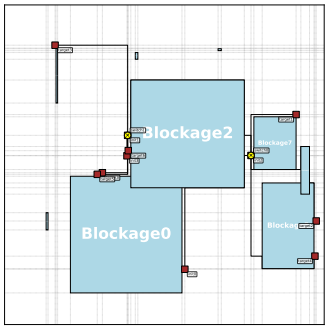
Random search



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.750, 0.650), (0.383, 0.485), (0.780, 0.650), (0.387, 0.592)

Active Route Paths:
init0 → switch0 → target0 ; init0 → switch0 → switch2 → target1
init0 → switch0 → target2 ; init0 → switch0 → switch3 → target5
init1 → switch3 → switch0 → target2 ; init1 → switch3 → switch0 → target4
init1 → switch3 → target6 ; init2 → switch0 → target0
init2 → switch0 → switch2 → target1 ; init2 → switch0 → target2
init2 → switch0 → target3 ; init2 → switch0 → target4
init2 → switch0 → switch3 → target6 ; init3 → switch1 → target0
init3 → switch3 → switch0 → target1 ; init3 → switch0 → target3
init3 → switch0 → target4 ; init3 → switch3 → target6

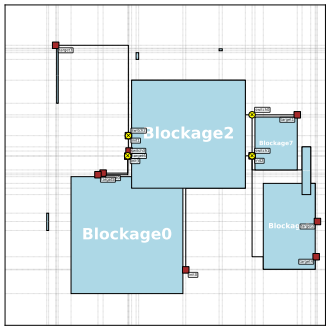
Genetic algorithm



NETLIST SUMMARY
Total Switches: 2
Switch Positions:
(0.771, 0.529), (0.385, 0.592)

Active Route Paths:
init0 → switch0 → switch1 → target0 ; init0 → switch0 → target1
init0 → switch0 → target2 ; init0 → switch0 → switch1 → target5
init1 → switch1 → switch0 → target2 ; init1 → switch1 → switch0 → target4
init1 → switch1 → target6 ; init2 → switch0 → switch1 → target0
init2 → switch0 → target1 ; init2 → switch0 → target2
init2 → switch0 → switch1 → target3 ; init2 → switch0 → target4
init2 → switch0 → switch1 → target6 ; init3 → switch1 → target0
init3 → switch1 → switch0 → target1 ; init3 → switch1 → target3
init3 → switch1 → switch0 → target4 ; init3 → switch1 → target6

PPO

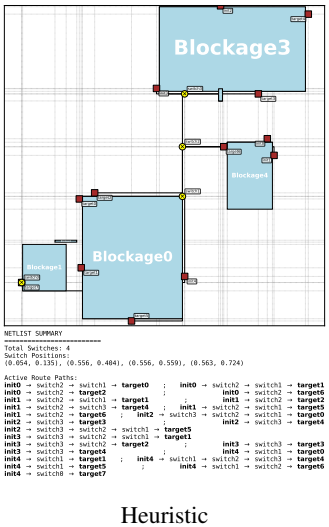


NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.771, 0.657), (0.771, 0.529), (0.385, 0.592), (0.383, 0.529)

Active Route Paths:
init0 → switch1 → switch0 → switch2 → switch3 → target0
init0 → switch1 → switch0 → target1 init0 → switch1 → target2
init0 → switch1 → switch0 → switch2 → switch3 → target5
init1 → switch2 → switch0 → switch1 → target2
init1 → switch2 → switch0 → switch1 → target4
init1 → switch2 → switch3 → target6
init2 → switch1 → switch0 → switch2 → switch3 → target0
init2 → switch1 → switch0 → target1 ; init2 → switch1 → target2
init2 → switch1 → switch0 → switch2 → target3 ; init2 → switch1 → target4
init2 → switch1 → switch0 → switch2 → switch3 → target6
init3 → switch3 → target0 ; init3 → switch3 → switch2 → switch0 → target1
init3 → switch3 → switch2 → target3
init3 → switch3 → switch2 → switch0 → switch1 → target4
init3 → switch3 → target6

MCTS

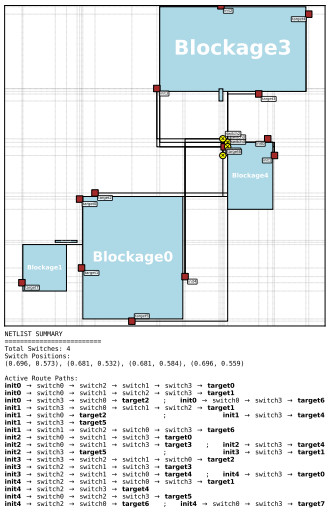
Figure 7: Instance 2.



NETLIST SUMMARY
==========
Total Switches: 4
Switch Positions:
(0.054, 0.135), (0.556, 0.404), (0.556, 0.559), (0.563, 0.724)

Active Route Paths:
init0 → switch2 → switch1 → **target0** ; **init0** → switch2 → switch1 → **target1**
init0 → switch2 → **target2** ; **init0** → switch2 → **target6**
init1 → switch2 → switch1 → **target1** ; **init1** → switch2 → **target2**
init1 → switch2 → switch3 → **target4** ; **init1** → switch2 → switch1 → **target5**
init1 → switch2 → **target6** ; **init2** → switch3 → switch2 → switch1 → **target0**
init2 → switch3 → **target3** ; **init2** → switch3 → **target4**
init2 → switch3 → switch2 → switch1 → **target5**
init3 → switch3 → switch2 → switch1 → **target1**
init3 → switch3 → switch2 → **target2** ; **init3** → switch3 → **target3**
init3 → switch3 → **target4** ; **init4** → switch3 → **target0**
init4 → switch1 → **target1** ; **init4** → switch1 → switch2 → switch3 → **target4**
init4 → switch1 → **target5** ; **init4** → switch1 → switch2 → **target6**
init4 → switch0 → **target7**

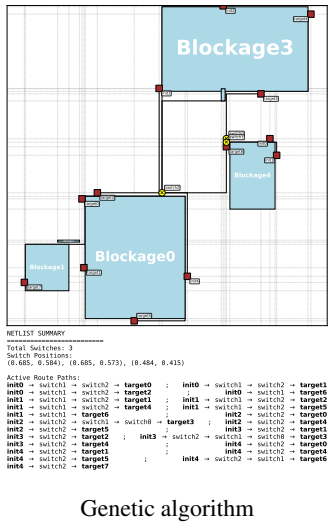
Heuristic



NETLIST SUMMARY
==========
Total Switches: 4
Switch Positions:
(0.696, 0.573), (0.681, 0.532), (0.681, 0.584), (0.696, 0.559)

Active Route Paths:
init0 → switch0 → switch2 → switch1 → switch3 → **target0**
init0 → switch0 → switch1 → switch2 → switch3 → **target1**
init0 → switch3 → switch0 → **target2** ; **init0** → switch0 → switch3 → **target6**
init1 → switch3 → switch0 → switch2 → switch1 → **target1**
init1 → switch0 → **target2** ; **init1** → switch3 → **target4**
init1 → switch3 → **target5**
init1 → switch1 → switch2 → switch0 → switch3 → **target6**
init2 → switch0 → switch1 → switch3 → **target0**
init2 → switch0 → switch1 → switch3 → **target3** ; **init2** → switch3 → **target4**
init2 → switch3 → **target5** ; **init3** → switch3 → **target1**
init3 → switch2 → switch1 → switch0 → **target2**
init3 → switch2 → switch1 → switch3 → **target3**
init3 → switch2 → switch1 → switch0 → **target4** ; **init4** → switch3 → **target0**
init4 → switch2 → switch1 → switch0 → switch3 → **target1**
init4 → switch2 → switch3 → **target4**
init4 → switch0 → switch2 → switch3 → **target5**
init4 → switch2 → switch0 → **target6** ; **init4** → switch0 → switch3 → **target7**

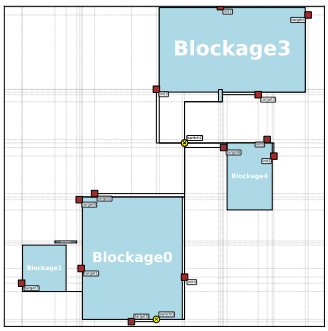
Random search



NETLIST SUMMARY
==========
Total Switches: 3
Switch Positions:
(0.685, 0.584), (0.685, 0.573), (0.484, 0.415)

Active Route Paths:
init0 → switch1 → switch2 → **target0** ; **init0** → switch1 → switch2 → **target1**
init0 → switch1 → switch2 → **target2** ; **init0** → switch1 → **target6**
init1 → switch1 → switch2 → **target1** ; **init1** → switch1 → switch2 → **target2**
init1 → switch1 → switch2 → **target4** ; **init1** → switch1 → switch2 → **target5**
init1 → switch1 → **target6** ; **init2** → switch2 → **target0**
init2 → switch2 → switch1 → switch0 → **target3** ; **init2** → switch2 → **target4**
init2 → switch2 → **target5** ; **init3** → switch2 → **target1**
init3 → switch2 → **target2** ; **init3** → switch2 → switch1 → switch0 → **target3**
init3 → switch2 → **target4** ; **init4** → switch2 → **target0**
init4 → switch2 → **target1** ; **init4** → switch2 → **target4**
init4 → switch2 → **target5** ; **init4** → switch2 → switch1 → **target6**
init4 → switch2 → **target7**

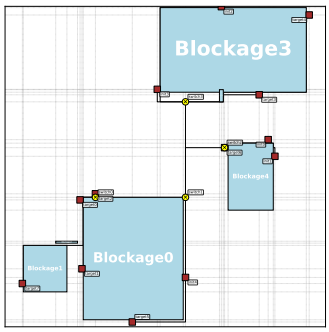
Genetic algorithm



NETLIST SUMMARY
==========
Total Switches: 3
Switch Positions:
(0.475, 0.022), (0.563, 0.573), (0.563, 0.573)

Active Route Paths:
init0 → switch2 → **target0** ; **init0** → switch2 → switch1 → **target1**
init0 → switch2 → **target2** ; **init0** → switch2 → **target6**
init1 → switch2 → switch1 → **target1** ; **init1** → switch2 → **target2**
init1 → switch2 → **target6** ; **init2** → switch2 → **target0**
init2 → switch2 → **target3** ; **init2** → switch2 → switch1 → **target4**
init2 → switch2 → switch0 → **target5** ; **init3** → switch2 → switch1 → **target1**
init3 → switch2 → **target2** ; **init3** → switch2 → **target3**
init3 → switch2 → switch1 → **target4** ; **init4** → switch2 → switch1 → **target0**
init4 → switch1 → **target1** ; **init4** → switch1 → switch2 → **target6**
init4 → switch0 → **target7**

PPO

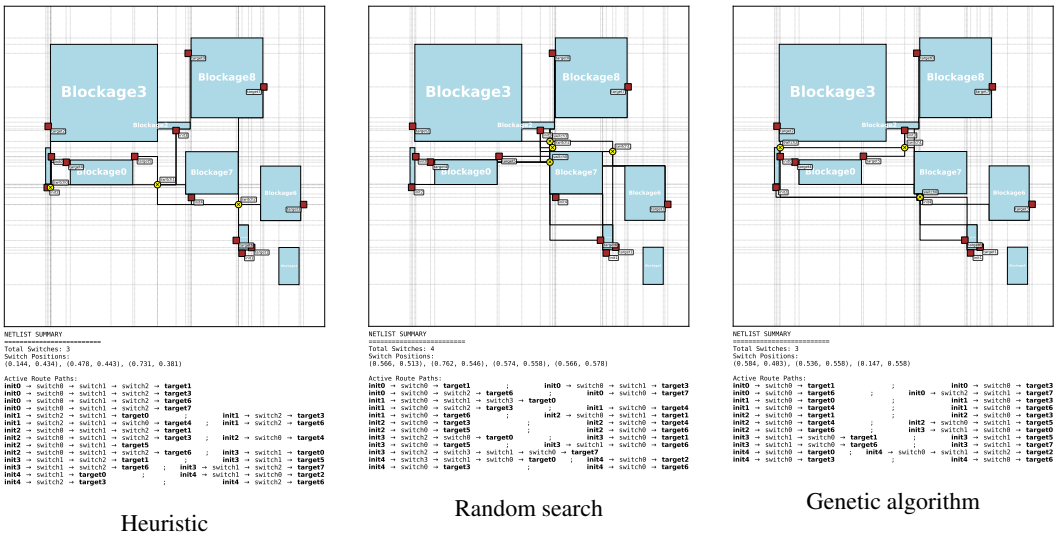


NETLIST SUMMARY
==========
Total Switches: 4
Switch Positions:
(0.282, 0.404), (0.563, 0.404), (0.685, 0.559), (0.563, 0.702)

Active Route Paths:
init0 → switch2 → switch1 → switch0 → **target0**
init0 → switch2 → switch1 → switch0 → **target1**
init0 → switch2 → switch1 → switch0 → **target2** ; **init0** → switch2 → **target6**
init1 → switch2 → switch1 → switch0 → **target1**
init1 → switch2 → switch1 → switch0 → **target2**
init1 → switch2 → switch1 → switch3 → **target4**
init1 → switch2 → switch1 → **target5** ; **init1** → switch2 → **target6**
init2 → switch3 → switch1 → switch0 → **target0** ; **init2** → switch3 → **target3**
init2 → switch3 → **target4** ; **init3** → switch3 → switch1 → **target5**
init3 → switch3 → switch1 → switch0 → **target2** ; **init3** → switch3 → **target3**
init3 → switch3 → **target4** ; **init4** → switch1 → switch0 → **target0**
init4 → switch1 → switch0 → **target1** ; **init4** → switch1 → switch3 → **target4**
init4 → switch1 → **target5** ; **init4** → switch1 → switch2 → **target6**
init4 → switch1 → switch0 → **target7**

MCTS

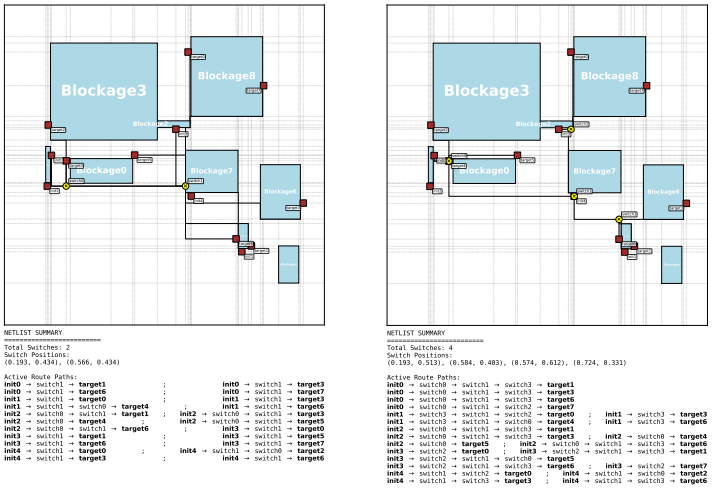
Figure 8: Instance 3.



Heuristic

Random search

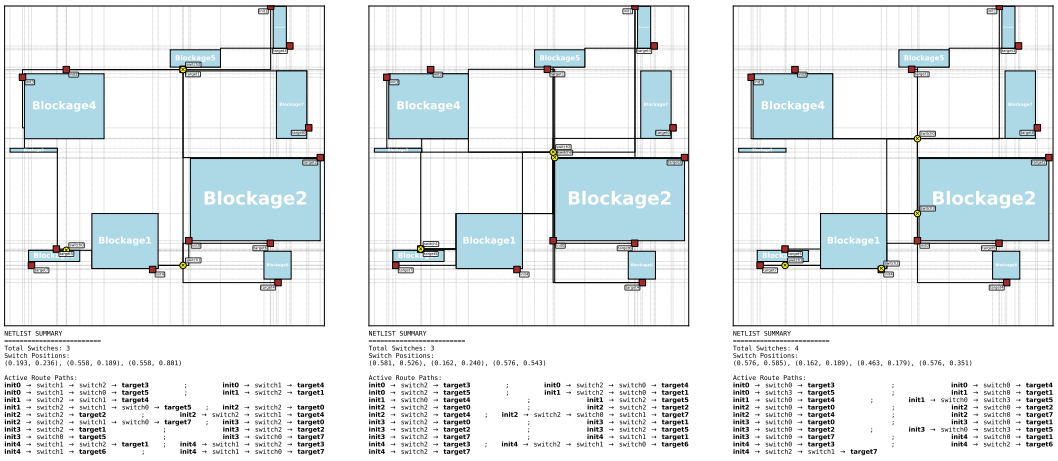
Genetic algorithm



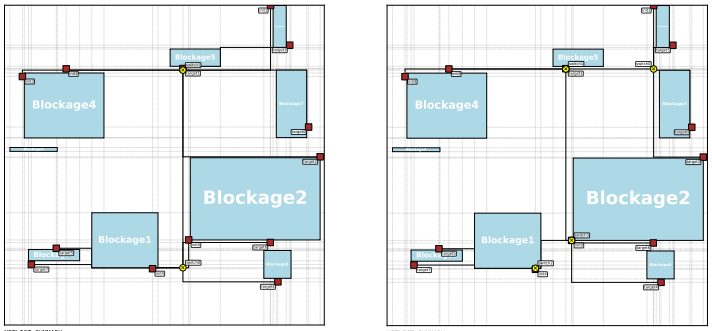
PPO

MCTS

Figure 9: Instance 4.

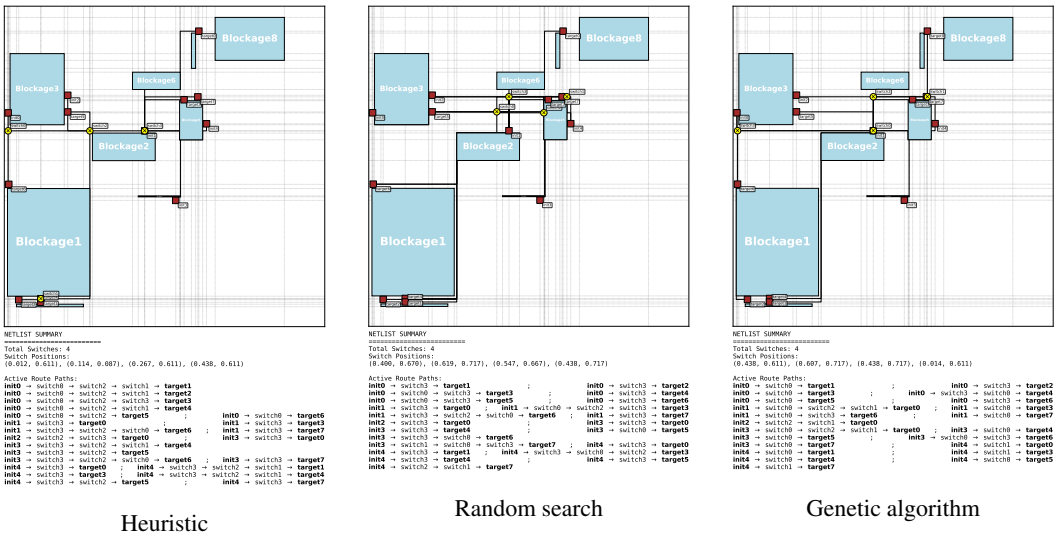


Heuristic Random search Genetic algorithm



PPO MCTS

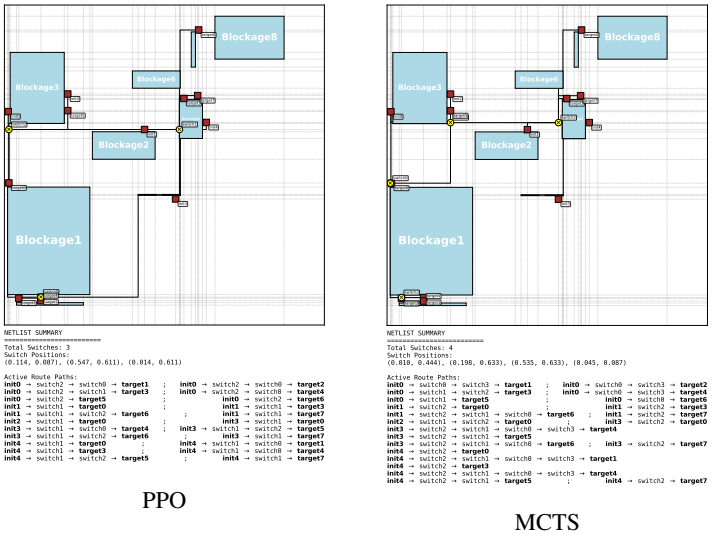
Figure 10: Instance 5.



Heuristic

Random search

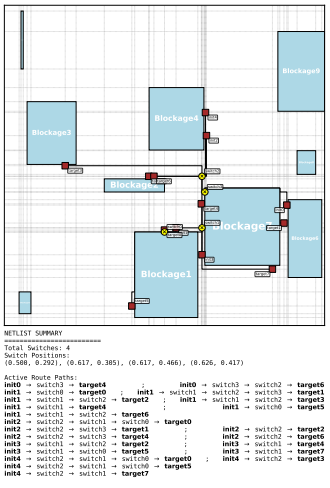
Genetic algorithm



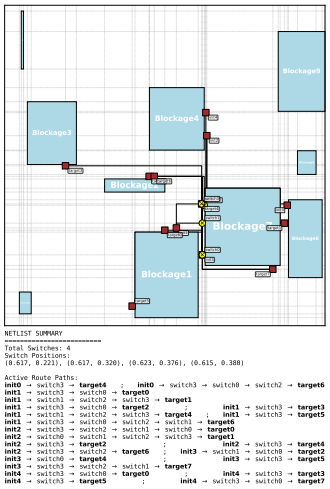
PPO

MCTS

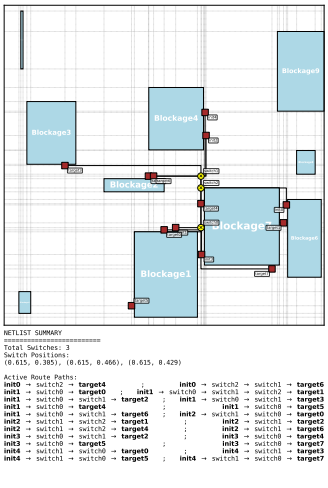
Figure 11: Instance 6.



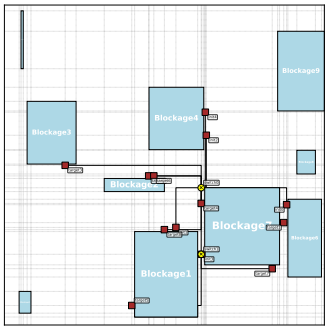
Heuristic



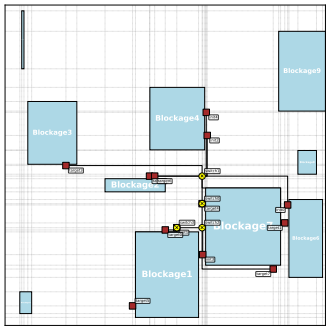
Random search



Genetic algorithm

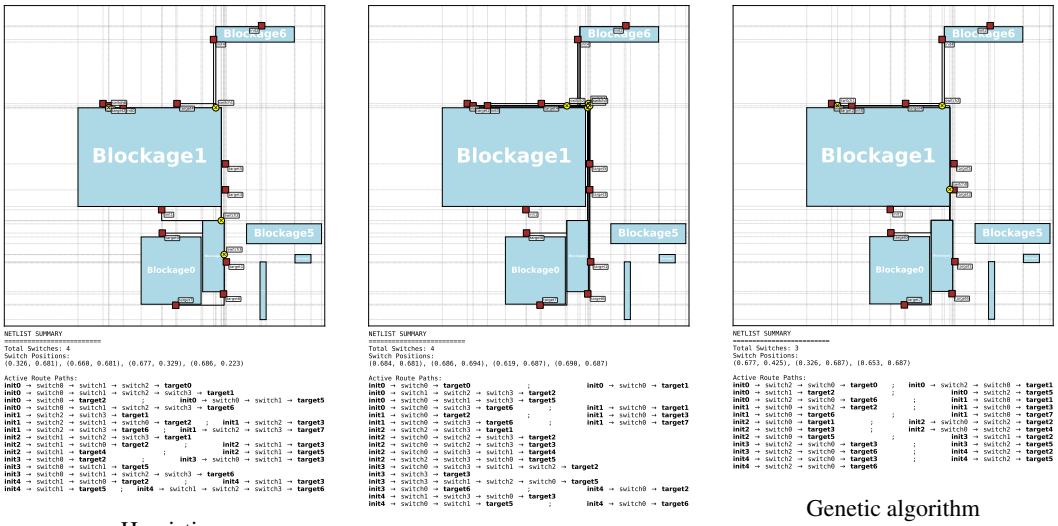


PPO



MCTS

Figure 12: Instance 7.



Heuristic

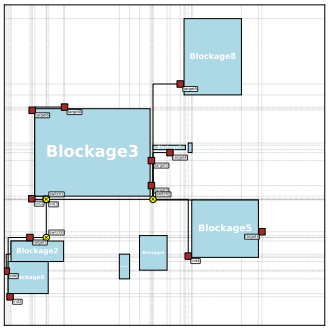
Random search

Genetic algorithm

PPO

MCTS

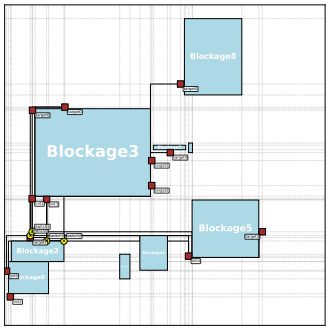
Figure 13: Instance 8.



NETLIST SUMMARY
Total Switches: 3
Switch Positions:
(0.131, 0.274), (0.131, 0.393), (0.464, 0.393)

Active Route Paths:
init0 → switch8 → switch1 → switch2 → **target2**
init0 → switch8 → switch1 → switch2 → **target3**
init0 → switch8 → switch1 → switch2 → **target4**
init0 → switch8 → switch1 → switch2 → **target5** ; **init0** → switch0 → **target7**
init1 → switch1 → switch2 → **target3** ; **init1** → switch1 → switch0 → **target7**
init2 → switch8 → switch1 → **target0**
init2 → switch8 → switch1 → switch2 → **target1**
init2 → switch8 → switch1 → switch2 → **target2**
init2 → switch8 → switch1 → switch2 → **target3**
init2 → switch8 → switch1 → switch2 → **target5** ; **init3** → switch1 → **target0**
init3 → switch1 → switch2 → **target1** ; **init3** → switch1 → switch2 → **target4**
init3 → switch1 → switch2 → **target5** ; **init3** → switch1 → **target6**
init4 → switch2 → **target1** ; **init4** → switch2 → **target5**
init4 → switch2 → switch1 → **target6**
init4 → switch2 → switch1 → switch0 → **target7**

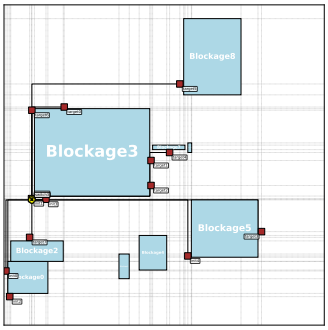
Heuristic



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.086, 0.280), (0.086, 0.292), (0.185, 0.263), (0.131, 0.263)

Active Route Paths:
init0 → switch8 → switch2 → **target2** ; **init0** → switch0 → switch1 → **target3**
init0 → switch8 → **target4** ; **init0** → switch0 → switch1 → switch3 → **target5**
init0 → switch8 → switch3 → **target7** ; **init1** → switch3 → switch1 → **target3**
init1 → switch3 → **target7** ; **init2** → switch3 → switch1 → **target0**
init2 → switch3 → **target1** ; **init3** → switch2 → switch0 → **target4**
init2 → switch3 → switch8 → switch1 → switch2 → **target2**
init2 → switch3 → switch8 → switch1 → switch2 → **target3**
init2 → switch3 → switch1 → switch2 → **target5** ; **init3** → switch3 → **target0**
init3 → switch3 → **target1** ; **init3** → switch3 → **target4**
init3 → switch2 → switch3 → **target6**
init4 → switch3 → switch2 → switch1 → **target3**
init4 → switch8 → switch3 → **target5** ; **init4** → switch0 → **target6**
init4 → switch0 → **target7**

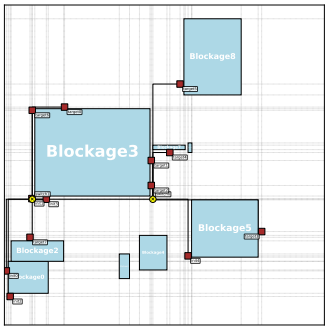
Random search



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.087, 0.391), (0.087, 0.395), (0.087, 0.391)

Active Route Paths:
init0 → switch8 → **target2** ; **init0** → switch8 → **target3**
init0 → switch8 → **target4** ; **init0** → switch8 → **target5**
init0 → switch8 → **target7**
init1 → switch2 → switch3 → switch1 → switch8 → **target3**
init1 → switch2 → switch3 → switch1 → switch8 → **target7**
init2 → switch8 → switch3 → switch1 → switch2 → **target0**
init2 → switch8 → **target1** ; **init2** → switch8 → **target2**
init2 → switch8 → **target3** ; **init2** → switch8 → **target5**
init3 → switch8 → switch1 → switch3 → switch2 → **target0**
init3 → switch1 → switch8 → **target1** ; **init3** → switch1 → switch8 → **target4**
init3 → switch8 → **target5** ; **init3** → switch8 → **target6**
init4 → switch1 → switch8 → **target3** ; **init4** → switch1 → switch8 → **target5**
init4 → switch1 → switch8 → **target6** ; **init4** → switch1 → switch8 → **target7**

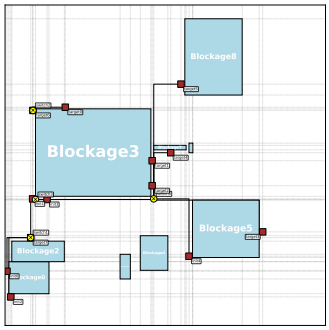
Genetic algorithm



NETLIST SUMMARY
Total Switches: 2
Switch Positions:
(0.464, 0.393), (0.087, 0.393)

Active Route Paths:
init0 → switch1 → switch0 → **target2** ; **init0** → switch1 → switch0 → **target3**
init0 → switch1 → switch0 → **target4** ; **init0** → switch1 → switch0 → **target5**
init0 → switch1 → **target7** ; **init1** → switch1 → switch0 → **target3**
init1 → switch1 → **target7** ; **init2** → switch1 → switch0 → **target0**
init2 → switch1 → switch0 → **target1** ; **init2** → switch1 → switch0 → **target2**
init2 → switch1 → switch0 → **target3** ; **init2** → switch1 → switch0 → **target5**
init3 → switch1 → **target0** ; **init3** → switch1 → switch0 → **target1**
init3 → switch1 → switch0 → **target4** ; **init3** → switch1 → switch0 → **target5**
init3 → switch1 → **target6** ; **init4** → switch0 → switch1 → **target3**
init4 → switch0 → **target5** ; **init4** → switch0 → switch1 → **target6**
init4 → switch0 → switch1 → **target7**

PPO

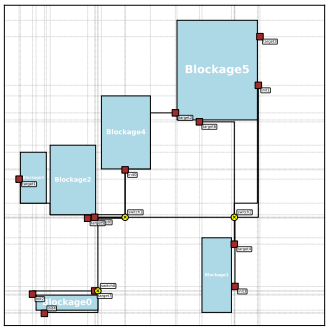


NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.464, 0.395), (0.880, 0.274), (0.087, 0.671), (0.095, 0.393)

Active Route Paths:
init0 → switch1 → switch3 → switch0 → **target2**
init0 → switch1 → switch3 → switch0 → **target3**
init0 → switch1 → switch3 → switch0 → **target4**
init0 → switch1 → switch3 → switch0 → **target5** ; **init0** → switch1 → **target7**
init1 → switch3 → switch0 → **target3** ; **init1** → switch3 → switch1 → **target7**
init2 → switch1 → switch3 → switch2 → **target0**
init2 → switch1 → switch3 → switch0 → **target1**
init2 → switch1 → switch3 → switch0 → **target2**
init2 → switch1 → switch3 → switch0 → **target3**
init2 → switch1 → switch3 → switch0 → **target5**
init3 → switch3 → switch2 → **target0** ; **init3** → switch3 → switch0 → **target1**
init3 → switch3 → switch0 → **target4** ; **init3** → switch3 → switch0 → **target5**
init3 → switch3 → switch2 → **target6** ; **init4** → switch0 → **target3**
init4 → switch0 → **target5** ; **init4** → switch0 → switch3 → switch2 → **target6**
init4 → switch0 → switch3 → switch1 → **target7**

MCTS

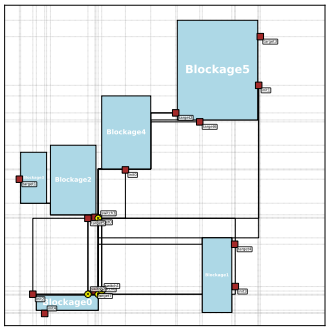
Figure 14: Instance 9.



NETLIST SUMMARY
Total Switches: 3
Switch Positions:
(0.292, 0.108), (0.377, 0.338), (0.718, 0.338)

Active Route Paths:
init0 → switch1 → target2 ; init0 → switch1 → switch2 → target3
init0 → switch1 → switch2 → target4 ; init1 → switch2 → switch1 → target1
init1 → switch2 → target3 ; init1 → switch2 → target4
init1 → switch2 → target6 ; init1 → switch2 → switch1 → switch0 → target7
init2 → switch0 → switch1 → target0 ; init2 → switch0 → switch1 → target1
init2 → switch0 → switch1 → target2
init2 → switch0 → switch1 → switch2 → target3
init2 → switch0 → switch1 → switch2 → target6 ; init2 → switch0 → target7
init3 → switch2 → switch1 → target1 ; init3 → switch2 → switch1 → target2
init3 → switch2 → target3 ; init3 → switch2 → switch1 → switch0 → target7
init4 → switch0 → switch1 → target1 ; init4 → switch0 → switch1 → target2
init4 → switch0 → switch1 → switch2 → target4
init4 → switch0 → switch1 → target5 ; init4 → switch0 → target7

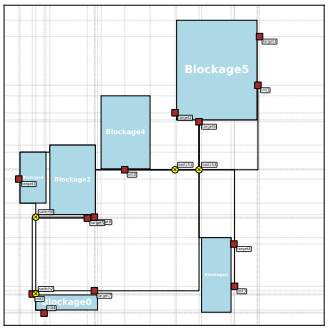
Heuristic



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.292, 0.098), (0.303, 0.108), (0.260, 0.098), (0.292, 0.335)

Active Route Paths:
init0 → switch3 → switch1 → switch0 → target2 ; init0 → switch3 → target3
init0 → switch3 → switch1 → switch0 → target4
init1 → switch2 → switch0 → switch3 → target1
init1 → switch2 → switch1 → switch0 → switch3 → target3 ; init1 → switch3 → target4
init1 → switch3 → target6 ; init1 → switch3 → target7
init2 → switch3 → switch1 → target0 ; init2 → switch3 → target1
init2 → switch3 → target2 ; init2 → switch3 → target3
init2 → switch3 → target5 ; init2 → switch2 → switch0 → switch1 → target7
init3 → switch0 → switch1 → switch2 → switch3 → target1
init3 → switch3 → target2 ; init3 → switch3 → target3
init3 → switch0 → switch2 → target7
init4 → switch3 → switch2 → switch1 → target1 ; init4 → switch3 → target2
init4 → switch2 → switch3 → target4
init4 → switch3 → switch1 → switch2 → target5
init4 → switch1 → switch0 → target7

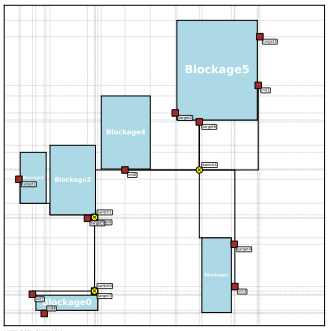
Random search



NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.899, 0.338), (0.534, 0.486), (0.099, 0.098), (0.609, 0.486)

Active Route Paths:
init0 → switch1 → switch3 → target2 ; init0 → switch1 → switch3 → target3
init0 → switch1 → switch3 → target4 ; init1 → switch3 → switch0 → target1
init1 → switch3 → target3 ; init1 → switch3 → target4
init1 → switch3 → target6 ; init1 → switch3 → target7
init2 → switch0 → target0 ; init2 → switch0 → target1
init2 → switch0 → switch3 → target2 ; init2 → switch0 → switch3 → target3
init2 → switch0 → switch1 → target6 ; init2 → switch2 → target7
init3 → switch3 → target1 ; init3 → switch3 → target2
init3 → switch3 → target3 ; init3 → switch3 → target7
init4 → switch0 → target1 ; init4 → switch0 → switch3 → target2
init4 → switch0 → switch3 → target4 ; init4 → switch0 → target5
init4 → switch2 → target7

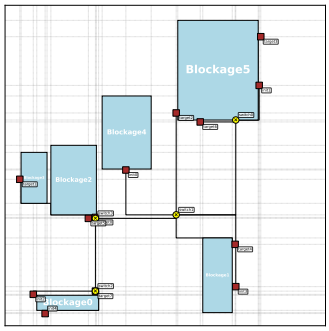
Genetic algorithm



NETLIST SUMMARY
Total Switches: 3
Switch Positions:
(0.282, 0.108), (0.282, 0.338), (0.609, 0.486)

Active Route Paths:
init0 → switch2 → target2 ; init0 → switch2 → target3
init0 → switch2 → target4 ; init1 → switch2 → switch1 → target1
init1 → switch2 → target3 ; init1 → switch2 → target4
init1 → switch2 → target6 ; init1 → switch2 → switch1 → switch0 → target7
init2 → switch0 → switch1 → target0 ; init2 → switch0 → switch1 → target1
init2 → switch0 → switch1 → target2
init2 → switch0 → switch1 → switch2 → target3
init2 → switch0 → switch1 → switch2 → target6 ; init2 → switch0 → target7
init3 → switch2 → switch1 → target1 ; init3 → switch2 → target2
init3 → switch2 → target3 ; init3 → switch2 → switch1 → switch0 → target7
init4 → switch0 → switch1 → target1
init4 → switch0 → switch1 → switch2 → target2
init4 → switch0 → switch1 → switch2 → target4
init4 → switch0 → switch1 → target5 ; init4 → switch0 → target7

PPO

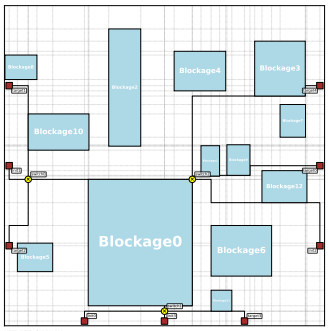


NETLIST SUMMARY
Total Switches: 4
Switch Positions:
(0.720, 0.642), (0.534, 0.346), (0.282, 0.108), (0.282, 0.335)

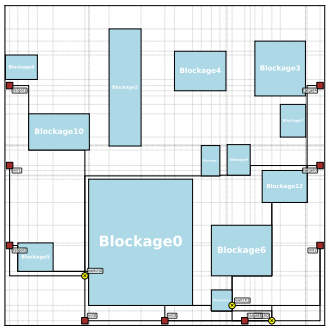
Active Route Paths:
init0 → switch1 → target2 ; init0 → switch1 → switch0 → target3
init0 → switch1 → target4 ; init1 → switch0 → switch1 → switch3 → target1
init1 → switch0 → target3 ; init1 → switch0 → switch1 → target4
init1 → switch0 → target6 ; init1 → switch0 → switch1 → switch3 → switch2 → target7
init2 → switch2 → switch3 → target0 ; init2 → switch2 → switch3 → target1
init2 → switch2 → switch3 → switch1 → target2
init2 → switch2 → switch3 → switch1 → switch0 → target3
init2 → switch2 → switch3 → switch1 → switch0 → target6
init2 → switch2 → target7 ; init3 → switch1 → switch3 → target1
init3 → switch1 → target2 ; init3 → switch1 → switch0 → target3
init3 → switch1 → switch3 → switch2 → target7
init4 → switch2 → switch3 → target1
init4 → switch2 → switch3 → switch1 → target2
init4 → switch2 → switch3 → switch1 → target4
init4 → switch2 → switch3 → target5 ; init4 → switch2 → target7

MCTS

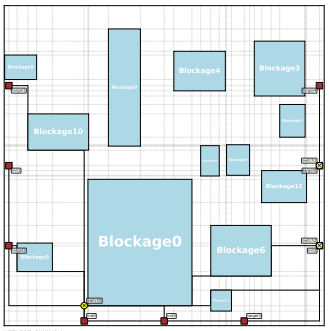
Figure 15: Instance 10.



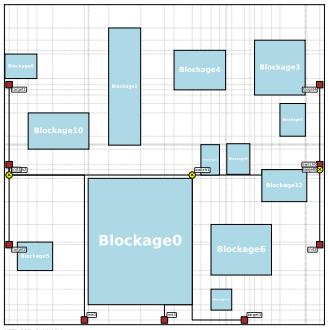
Heuristic



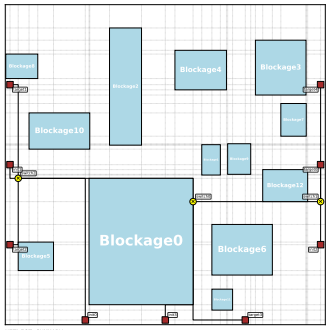
Random search



Genetic algorithm

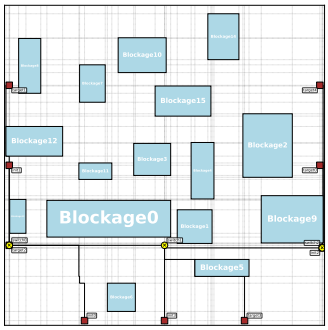


PPO

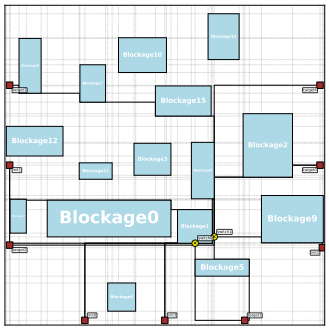


MCTS

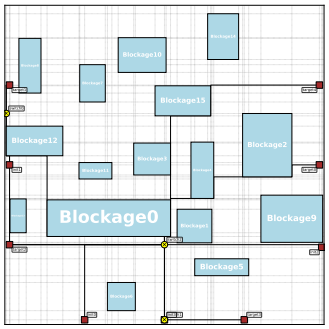
Figure 16: Instance 11.



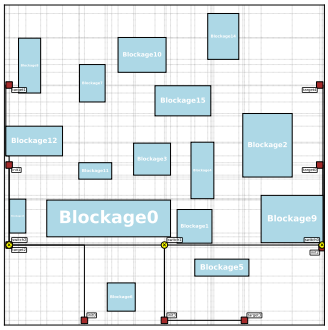
Heuristic



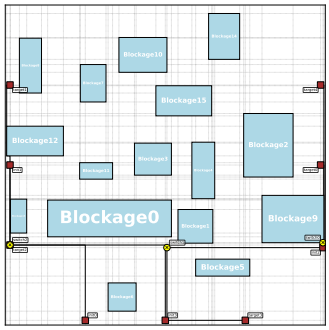
Random search



Genetic algorithm

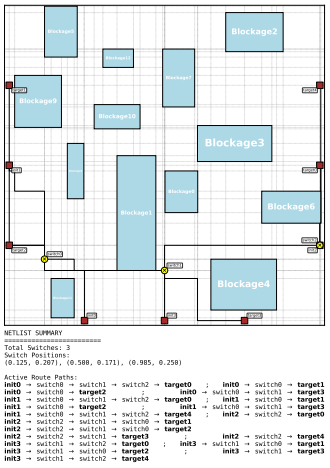


PPO

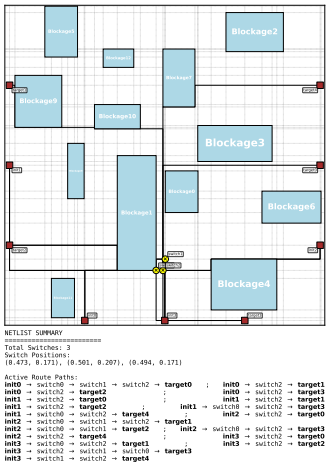


MCTS

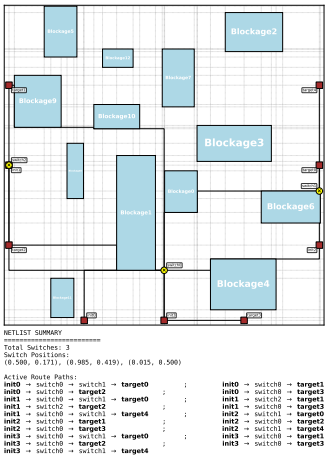
Figure 17: Instance 12.



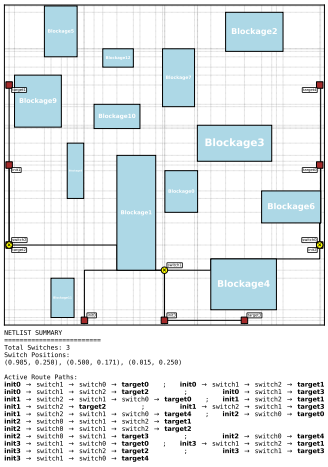
Heuristic



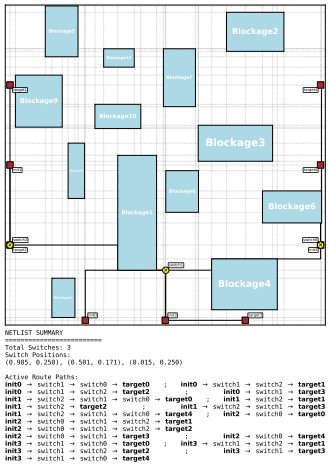
Random search



Genetic algorithm

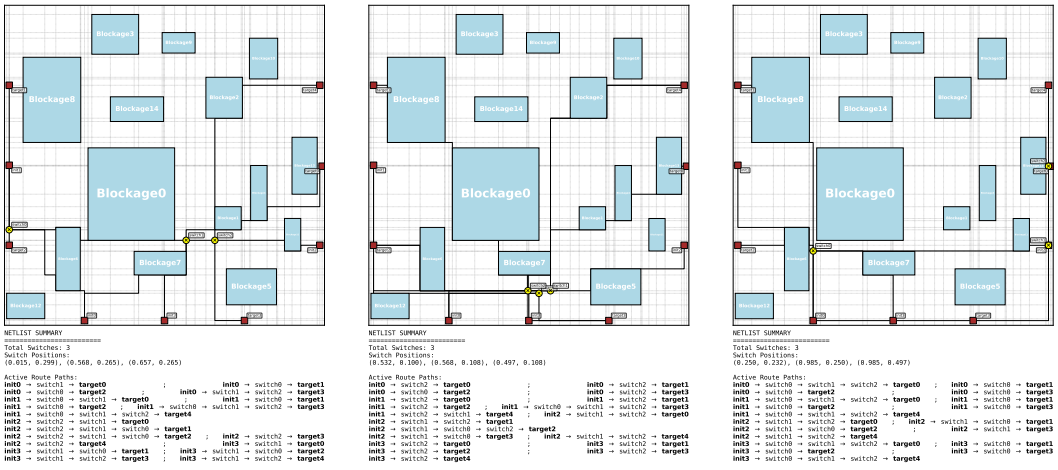


PPO

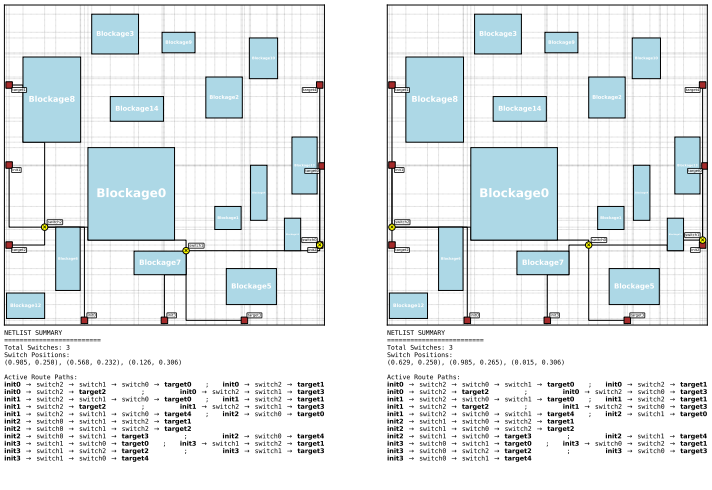


MCTS

Figure 18: Instance 13.

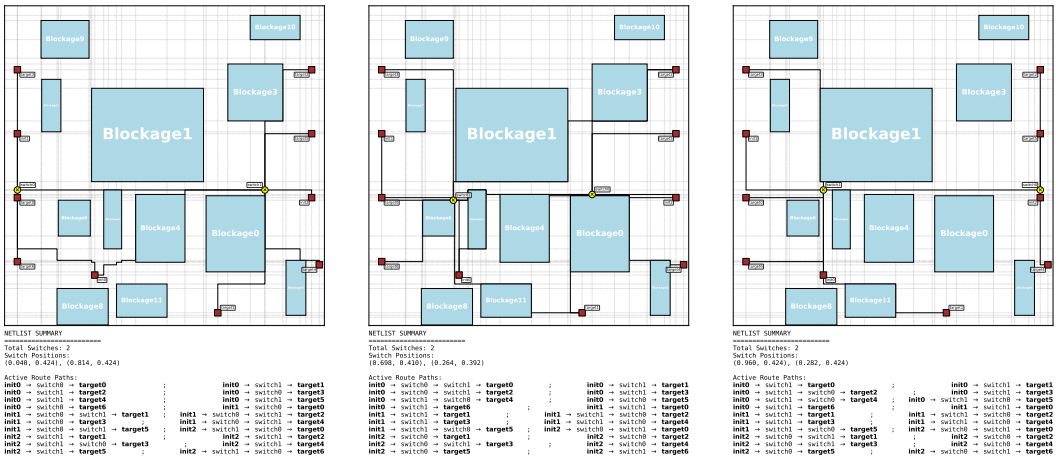


Heuristic Random search Genetic algorithm



PPO MCTS

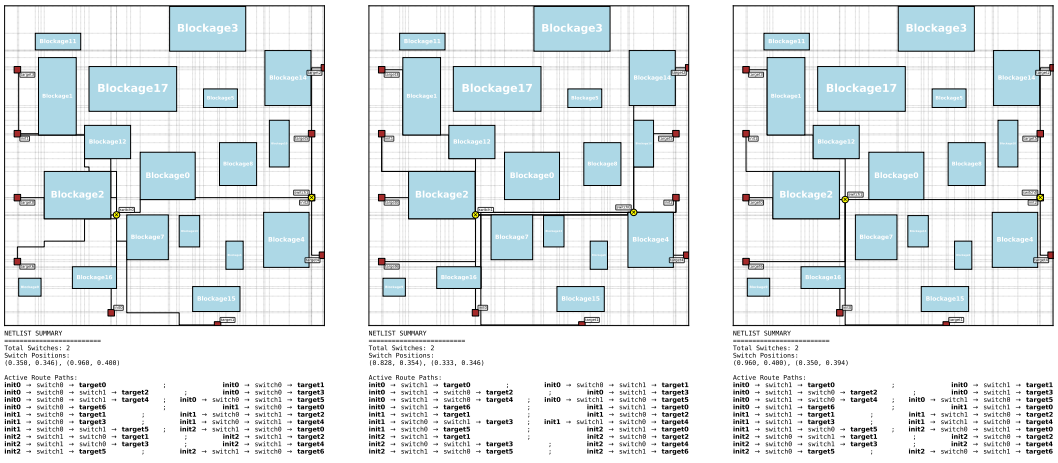
Figure 19: Instance 14.



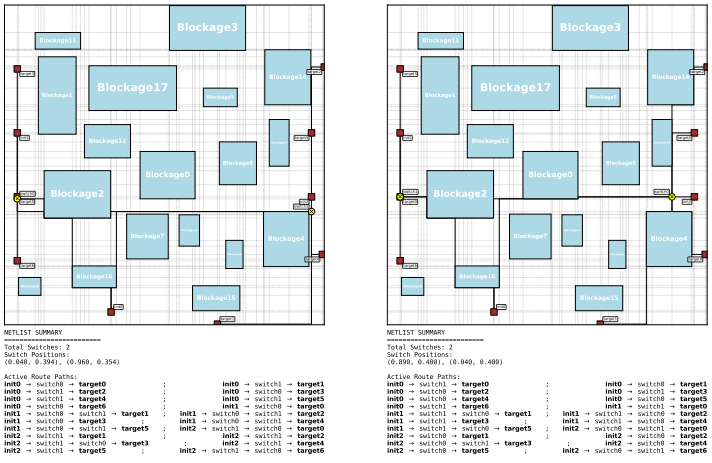
Heuristic Random search Genetic algorithm

PPO MCTS

Figure 20: Instance 15.

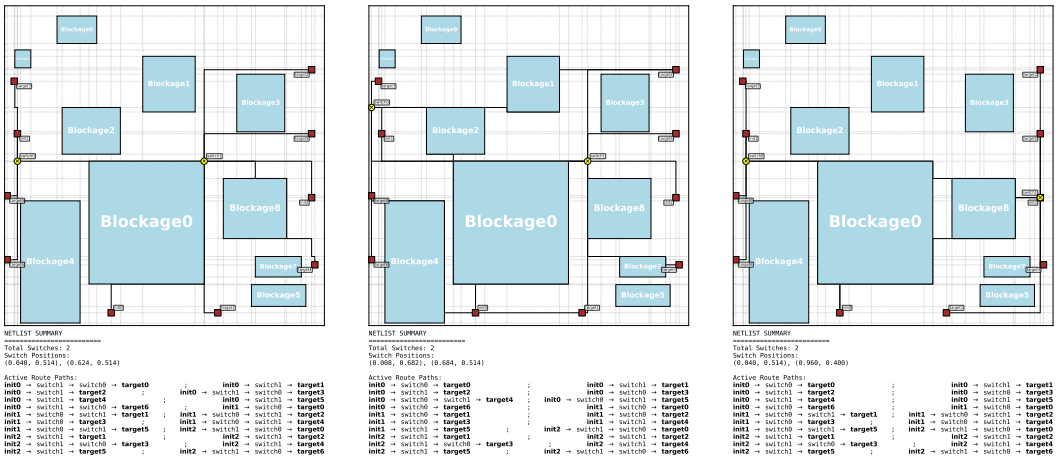


Heuristic Random search Genetic algorithm

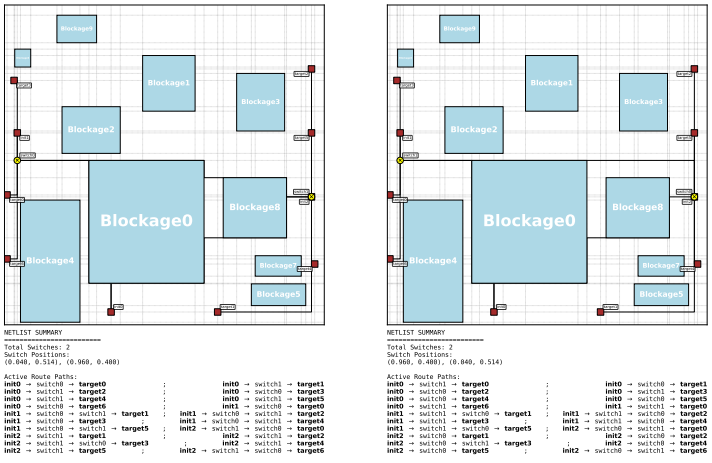


PPO MCTS

Figure 21: Instance 16.

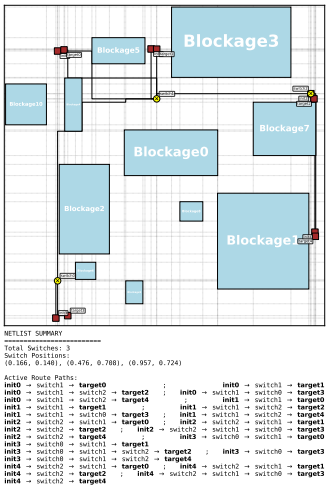


Heuristic Random search Genetic algorithm



PPO MCTS

Figure 22: Instance 17.



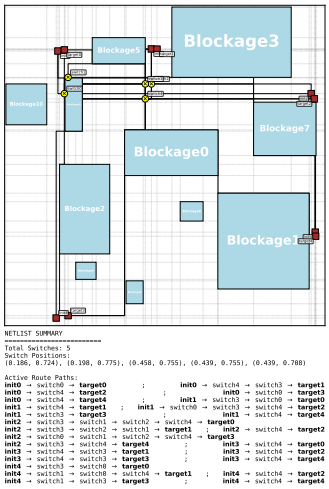
NETLIST SUMMARY

Total Switches: 3
Switch Positions:
(0.166, 0.348), (0.476, 0.788), (0.957, 0.724)

Active Route Paths:

init0 → switch1 → target0	;	init0 → switch1 → target1
init0 → switch1 → switch2 → target2	;	init0 → switch1 → switch0 → target3
init0 → switch1 → switch2 → target4		init1 → switch1 → switch2 → target0
init1 → switch1 → target1	;	init1 → switch1 → switch2 → target2
init1 → switch1 → switch0 → target3	;	init1 → switch1 → switch2 → target4
init2 → switch2 → switch1 → target0	;	init2 → switch2 → switch1 → target1
init2 → switch2 → target2	;	init2 → switch2 → switch1 → switch0 → target3
init2 → switch2 → target4	;	init3 → switch0 → switch1 → target0
init3 → switch0 → switch1 → target1		
init3 → switch0 → switch1 → switch2 → target2	;	init3 → switch0 → target3
init3 → switch0 → switch1 → switch2 → target4		
init4 → switch2 → switch1 → target0	;	init4 → switch2 → switch1 → target1
init4 → switch2 → target2	;	init4 → switch2 → switch1 → switch0 → target3
init4 → switch2 → target4		

Heuristic



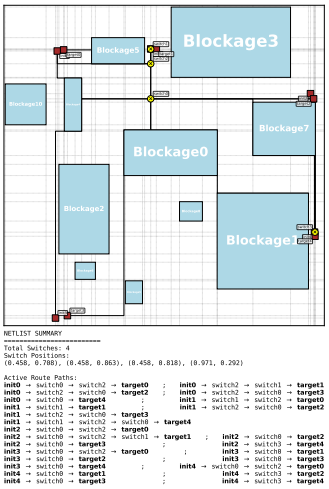
NETLIST SUMMARY

Total Switches: 5
Switch Positions:
(0.186, 0.724), (0.198, 0.775), (0.458, 0.755), (0.439, 0.755), (0.439, 0.708)

Active Route Paths:

init0 → switch0 → target0	;	init0 → switch4 → switch3 → target1
init0 → switch4 → target2	;	init0 → switch0 → switch3 → target3
init0 → switch4 → target4		init1 → switch3 → switch0 → target0
init1 → switch4 → target1	;	init1 → switch0 → switch3 → switch4 → target2
init1 → switch3 → target3	;	init1 → switch4 → target4
init2 → switch3 → switch1 → switch2 → switch4 → target0		
init2 → switch3 → switch2 → switch1 → target1	;	init2 → switch4 → target2
init2 → switch0 → switch1 → switch2 → switch4 → target3		
init2 → switch3 → switch4 → target4	;	init3 → switch4 → target0
init3 → switch4 → switch3 → target1	;	init3 → switch4 → target2
init3 → switch4 → switch3 → target3	;	init3 → switch4 → target4
init4 → switch3 → switch0 → target0		
init4 → switch1 → switch0 → switch4 → target1	;	init4 → switch4 → target2
init4 → switch1 → switch3 → target3	;	init4 → switch4 → target4

Random search



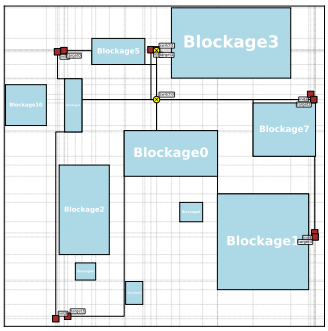
NETLIST SUMMARY

Total Switches: 4
Switch Positions:
(0.458, 0.780), (0.458, 0.863), (0.458, 0.818), (0.971, 0.292)

Active Route Paths:

init0 → switch2 → target0	;	init0 → switch2 → switch1 → target1
init0 → switch2 → switch0 → target2	;	init0 → switch2 → switch0 → target3
init0 → switch0 → target4	;	init1 → switch1 → switch2 → target0
init1 → switch1 → target1	;	init1 → switch2 → switch0 → target2
init1 → switch2 → switch0 → target3		
init1 → switch1 → switch2 → switch0 → target4		
init2 → switch0 → switch2 → target0		
init2 → switch0 → switch2 → switch1 → target1	;	init2 → switch0 → target2
init2 → switch0 → target3	;	init2 → switch3 → target4
init3 → switch0 → switch2 → target0	;	init3 → switch0 → target1
init3 → switch0 → target2	;	init3 → switch0 → target3
init3 → switch0 → target4	;	init4 → switch0 → switch2 → target0
init4 → switch0 → target1	;	init4 → switch3 → target2
init4 → switch0 → target3	;	init4 → switch3 → target4

Genetic algorithm



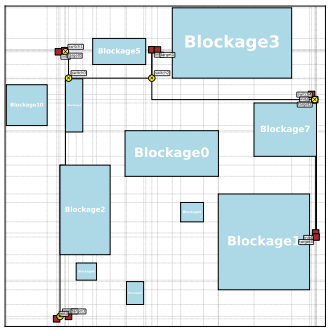
NETLIST SUMMARY

Total Switches: 2
Switch Positions:
(0.476, 0.788), (0.476, 0.861)

Active Route Paths:

init0 → switch0 → target0	;	init0 → switch0 → target1
init0 → switch0 → target2	;	init0 → switch0 → target3
init0 → switch0 → target4	;	init1 → switch0 → target0
init1 → switch1 → target1	;	init1 → switch0 → target2
init1 → switch0 → target3	;	init1 → switch0 → target4
init2 → switch0 → target0	;	init2 → switch0 → target1
init2 → switch0 → target2	;	init2 → switch0 → target3
init2 → switch0 → target4	;	init3 → switch0 → target0
init3 → switch0 → target1	;	init3 → switch0 → target2
init3 → switch0 → target3	;	init3 → switch0 → target4
init4 → switch0 → target0	;	init4 → switch0 → target1
init4 → switch0 → target2	;	init4 → switch0 → target3
init4 → switch0 → target4		

PPO



NETLIST SUMMARY

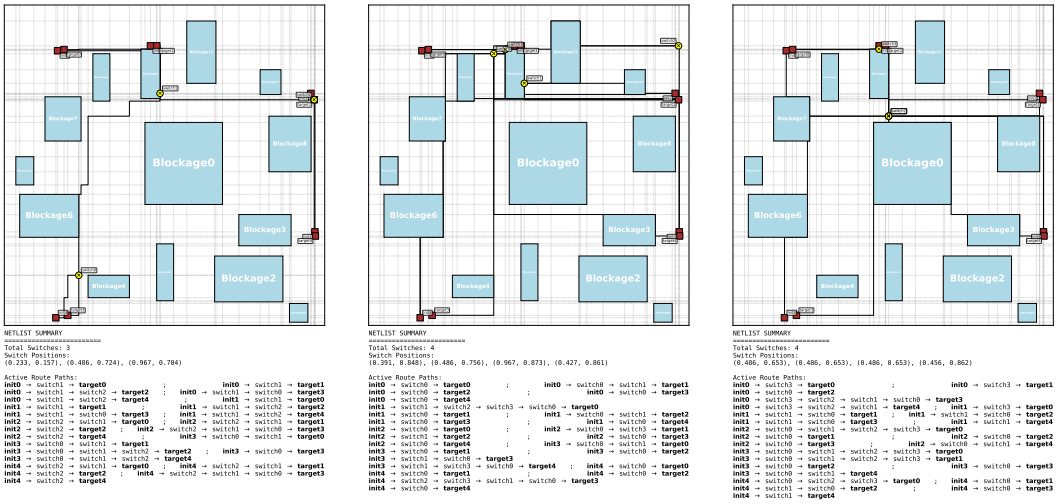
Total Switches: 5
Switch Positions:
(0.198, 0.775), (0.189, 0.857), (0.458, 0.775), (0.172, 0.031), (0.967, 0.708)

Active Route Paths:

init0 → switch1 → target0	;	init0 → switch1 → switch0 → switch2 → target1
init0 → switch1 → switch0 → switch2 → switch4 → target2		
init0 → switch1 → switch0 → switch3 → target3		
init0 → switch1 → switch0 → switch2 → switch4 → target4		
init1 → switch2 → switch1 → switch0 → target0	;	init1 → switch2 → target1
init1 → switch2 → switch4 → target2		
init1 → switch2 → switch0 → switch3 → target3		
init2 → switch4 → switch2 → switch0 → switch1 → target0		
init2 → switch4 → switch2 → target1	;	init2 → switch4 → target2
init2 → switch4 → target4	;	init3 → switch3 → switch0 → switch1 → target0
init3 → switch3 → switch0 → switch2 → target1		
init3 → switch3 → switch0 → switch2 → switch4 → target2		
init3 → switch3 → target3		
init4 → switch0 → switch2 → switch4 → target4		
init4 → switch4 → switch2 → switch0 → switch1 → target0		
init4 → switch4 → switch2 → target1	;	init4 → switch4 → target2
init4 → switch4 → switch2 → switch0 → switch3 → target3		
init4 → switch4 → target4		

MCTS

Figure 23: Instance 18.



Heuristic

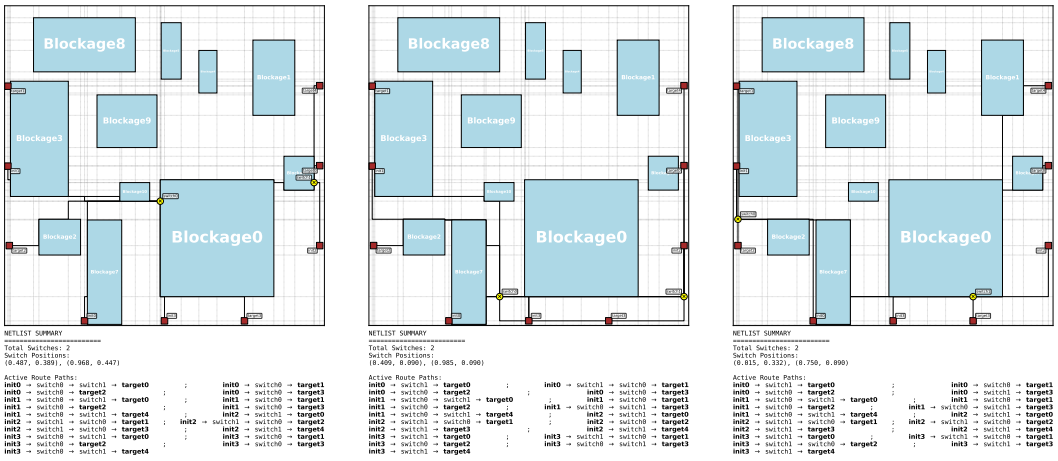
Random search

Genetic algorithm

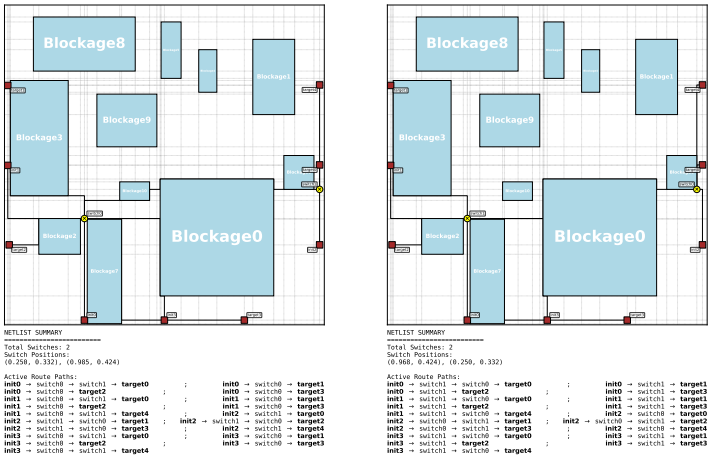
PPO

MCTS

Figure 24: Instance 19.

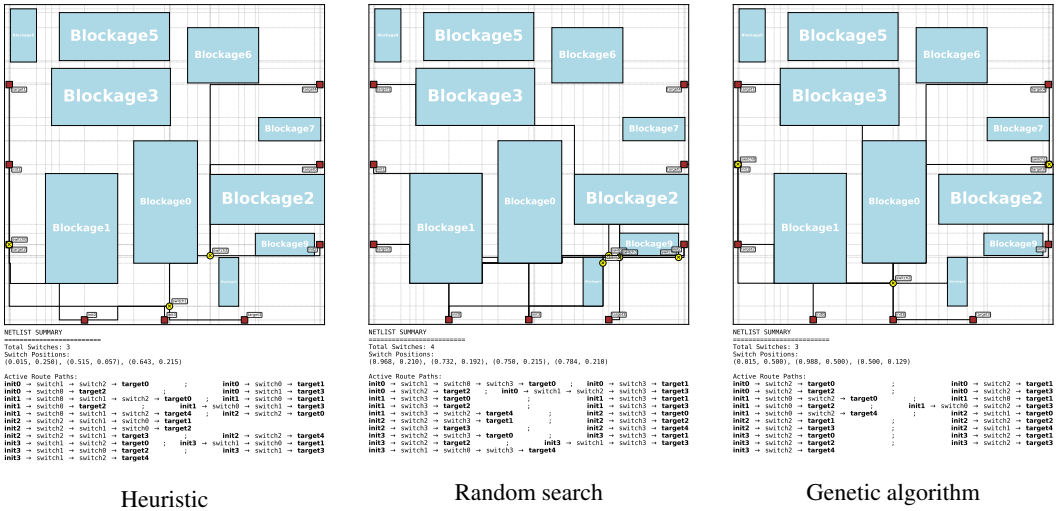


Heuristic Random search Genetic algorithm

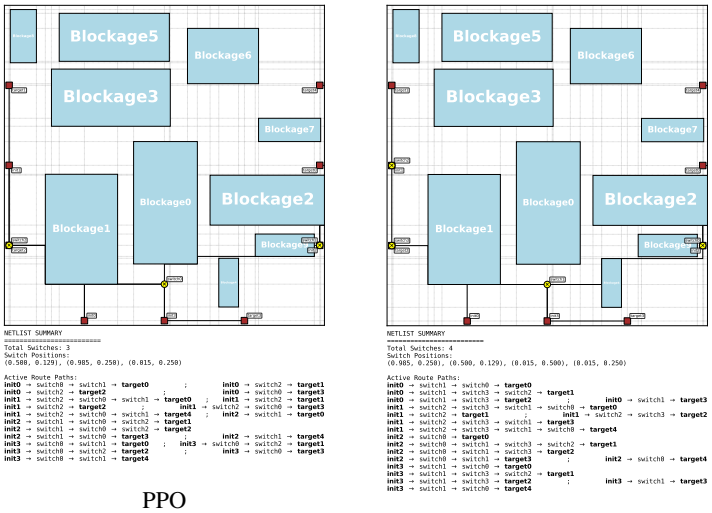


PPO MCTS

Figure 25: Instance 20.

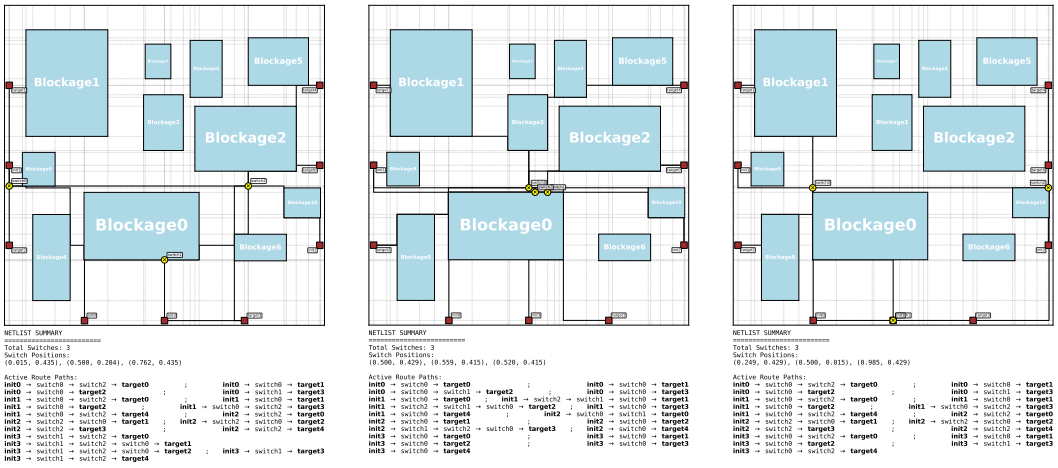


Heuristic Random search Genetic algorithm

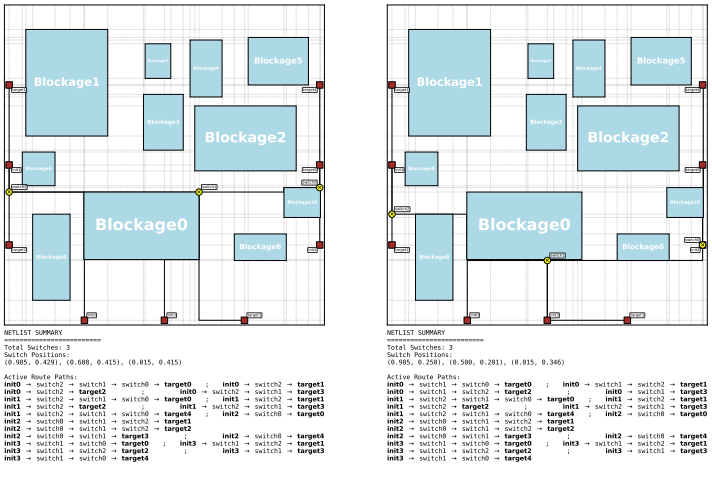


PPO MCTS

Figure 26: Instance 21.

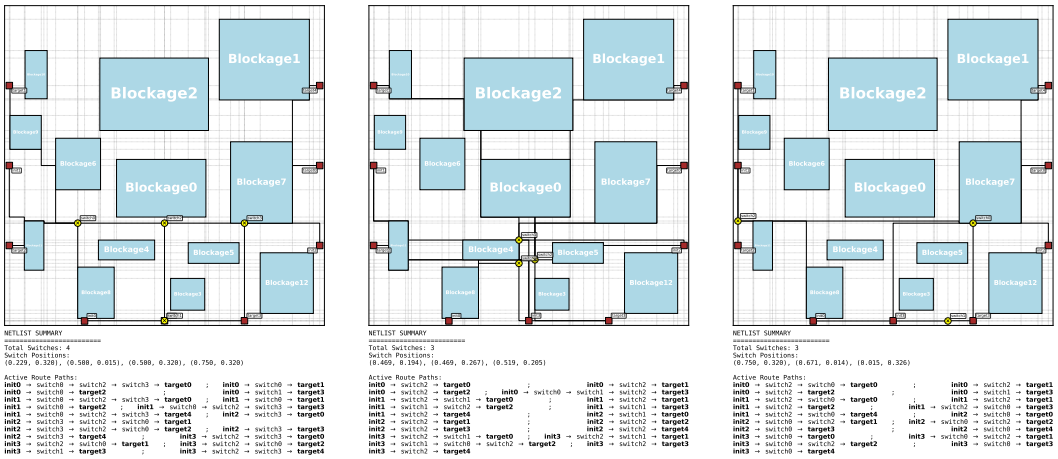


Heuristic Random search Genetic algorithm



PPO MCTS

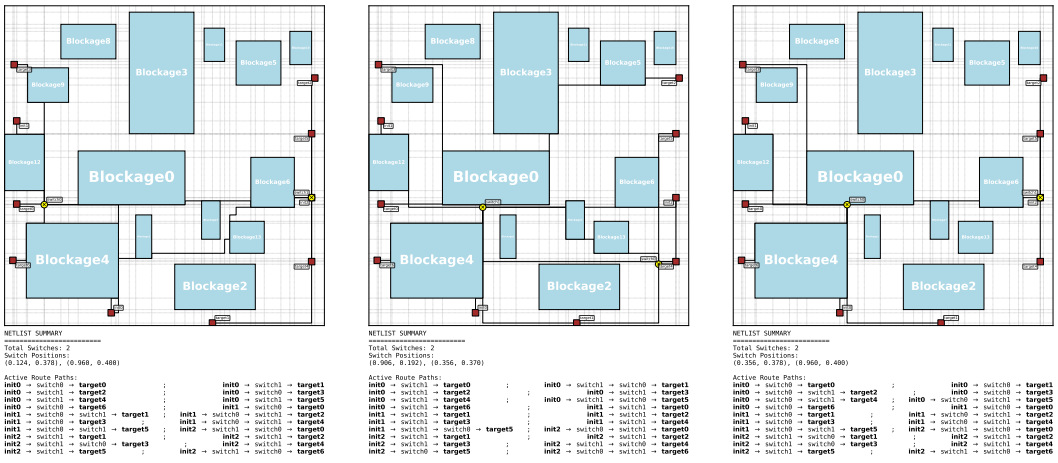
Figure 27: Instance 22.



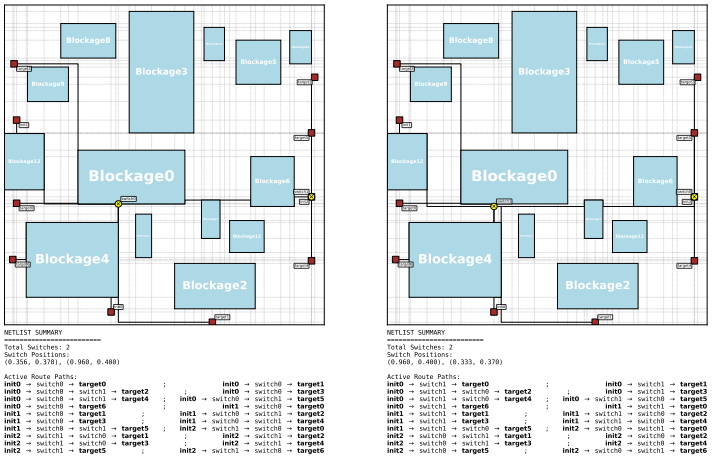
Heuristic Random search Genetic algorithm

PPO MCTS

Figure 28: Instance 23.



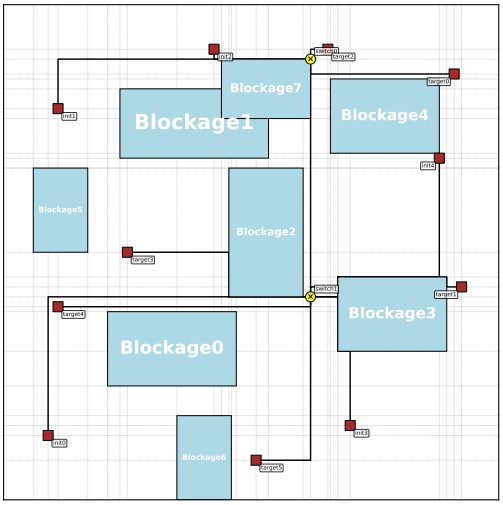
Heuristic Random search Genetic algorithm



PPO MCTS

Figure 29: Instance 24.

E.2 FINE-TUNING

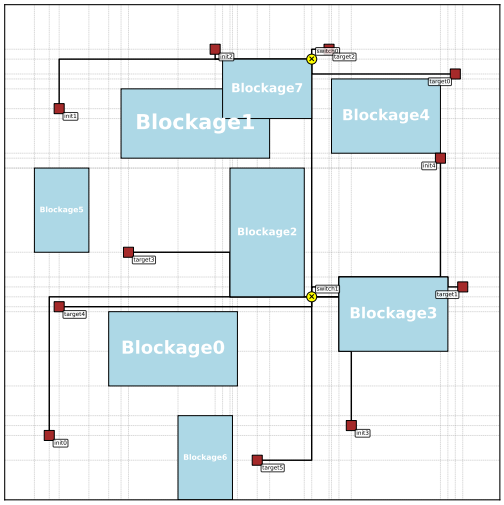


NETLIST SUMMARY
==========
Total Switches: 2
Switch Positions:
(0.620, 0.890), (0.620, 0.410)

Active Route Paths:

init0 → switch1 → switch0 → **target0**	;	**init0** → switch1 → **target1**
init0 → switch1 → **target3**	;	**init0** → switch1 → **target5**
init1 → switch0 → **target0**	;	**init1** → switch0 → **target2**
init1 → switch0 → switch1 → **target4**	;	**init1** → switch0 → switch1 → **target5**
init2 → switch0 → switch1 → **target1**	;	**init2** → switch0 → **target2**
init2 → switch0 → switch1 → **target3**	;	**init2** → switch0 → switch1 → **target5**
init3 → switch1 → switch0 → **target0**	;	**init3** → switch1 → switch0 → **target2**
init3 → switch1 → **target4**	;	**init3** → switch1 → **target5**
init4 → switch1 → **target1**	;	**init4** → switch1 → **target3**
init4 → switch1 → **target4**	;	**init4** → switch1 → **target5**

PPO, pretrained

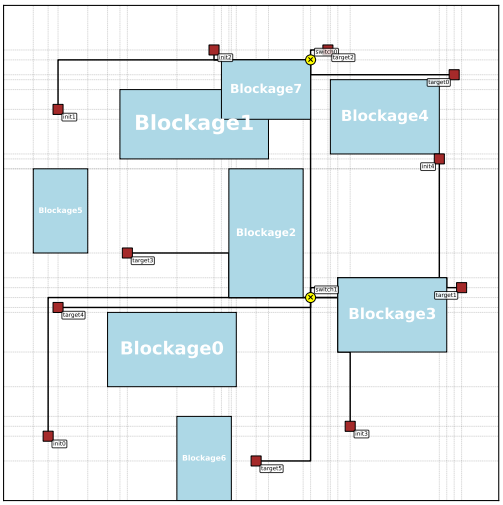


NETLIST SUMMARY
==========
Total Switches: 2
Switch Positions:
(0.620, 0.890), (0.620, 0.410)

Active Route Paths:

init0 → switch1 → switch0 → **target0**	;	**init0** → switch1 → **target1**
init0 → switch1 → **target3**	;	**init0** → switch1 → **target5**
init1 → switch0 → **target0**	;	**init1** → switch0 → **target2**
init1 → switch0 → switch1 → **target4**	;	**init1** → switch0 → switch1 → **target5**
init2 → switch0 → switch1 → **target1**	;	**init2** → switch0 → **target2**
init2 → switch0 → switch1 → **target3**	;	**init2** → switch0 → switch1 → **target5**
init3 → switch1 → switch0 → **target0**	;	**init3** → switch1 → switch0 → **target2**
init3 → switch1 → **target4**	;	**init3** → switch1 → **target5**
init4 → switch1 → **target1**	;	**init4** → switch1 → **target3**
init4 → switch1 → **target4**	;	**init4** → switch1 → **target5**

PPO, from scratch

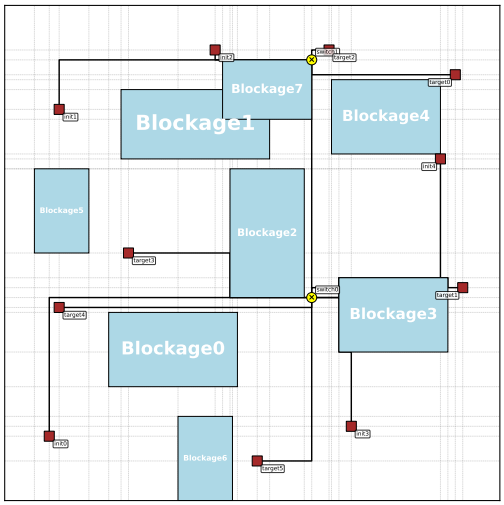


NETLIST SUMMARY
==========
Total Switches: 2
Switch Positions:
(0.620, 0.890), (0.620, 0.410)

Active Route Paths:

init0 → switch1 → switch0 → **target0**	;	**init0** → switch1 → **target1**
init0 → switch1 → **target3**	;	**init0** → switch1 → **target5**
init1 → switch0 → **target0**	;	**init1** → switch0 → **target2**
init1 → switch0 → switch1 → **target4**	;	**init1** → switch0 → switch1 → **target5**
init2 → switch0 → switch1 → **target1**	;	**init2** → switch0 → **target2**
init2 → switch0 → switch1 → **target3**	;	**init2** → switch0 → switch1 → **target5**
init3 → switch1 → switch0 → **target0**	;	**init3** → switch1 → switch0 → **target2**
init3 → switch1 → **target4**	;	**init3** → switch1 → **target5**
init4 → switch1 → **target1**	;	**init4** → switch1 → **target3**
init4 → switch1 → **target4**	;	**init4** → switch1 → **target5**

MCTS, pretrained



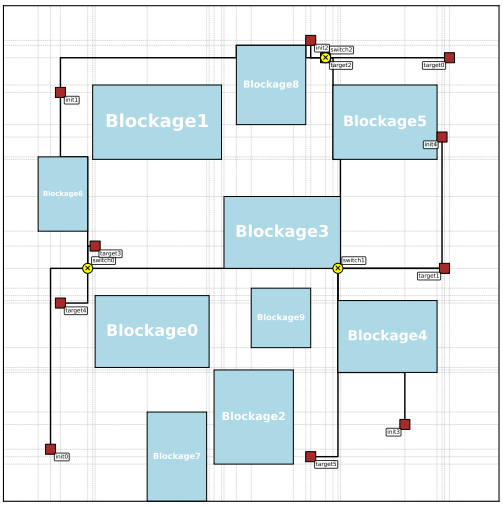
NETLIST SUMMARY
==========
Total Switches: 2
Switch Positions:
(0.620, 0.410), (0.620, 0.890)

Active Route Paths:

init0 → switch1 → switch0 → **target0**	;	**init0** → switch0 → **target1**
init0 → switch0 → **target3**	;	**init0** → switch0 → **target5**
init1 → switch1 → **target0**	;	**init1** → switch1 → **target2**
init1 → switch1 → switch0 → **target4**	;	**init1** → switch1 → switch0 → **target5**
init2 → switch1 → switch0 → **target1**	;	**init2** → switch1 → switch0 → **target2**
init2 → switch1 → switch0 → **target3**	;	**init2** → switch1 → switch0 → **target5**
init3 → switch0 → switch1 → **target0**	;	**init3** → switch0 → switch1 → **target2**
init3 → switch0 → **target4**	;	**init3** → switch0 → **target5**
init4 → switch0 → **target1**	;	**init4** → switch0 → **target3**
init4 → switch0 → **target4**	;	**init4** → switch0 → **target5**

MCTS, from scratch

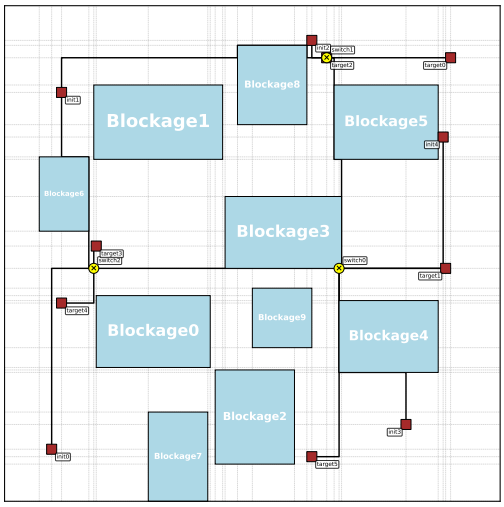
Figure 30: Fine-tuning instance 1.




```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.170, 0.470), (0.675, 0.470), (0.650, 0.895)

Active Route Paths:
init0 → switch0 → switch1 → switch2 → target0
init0 → switch0 → switch1 → target1          ;          init0 → switch0 → target3
init0 → switch0 → switch1 → target5          ;          init1 → switch2 → target0
init1 → switch2 → target2                    ;          init1 → switch0 → target4
init1 → switch0 → switch1 → target5    ;    init2 → switch2 → switch1 → target1
init2 → switch2 → target2    ;    init2 → switch2 → switch1 → switch0 → target3
init2 → switch2 → switch1 → target5    ;    init3 → switch1 → switch2 → target0
init3 → switch1 → switch2 → target2    ;    init3 → switch1 → switch0 → target4
init3 → switch1 → target5                    ;          init4 → switch1 → target1
init4 → switch1 → switch0 → target3    ;    init4 → switch1 → switch0 → target4
init4 → switch1 → target5
```

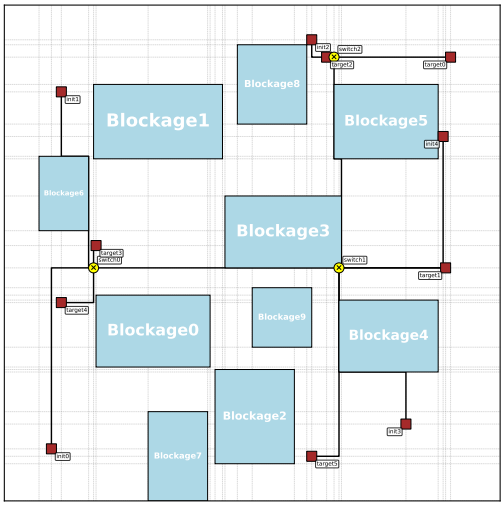
PPO, pretrained




```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.675, 0.470), (0.650, 0.895), (0.180, 0.470)

Active Route Paths:
init0 → switch2 → switch0 → switch1 → target0
init0 → switch2 → switch0 → target1          ;          init0 → switch2 → target3
init0 → switch2 → switch0 → target5          ;          init1 → switch1 → target0
init1 → switch1 → target2                    ;          init1 → switch2 → target4
init1 → switch2 → switch0 → target5    ;    init2 → switch1 → switch0 → target1
init2 → switch1 → target2    ;    init2 → switch1 → switch0 → switch2 → target3
init2 → switch1 → switch0 → target5    ;    init3 → switch0 → switch1 → target0
init3 → switch0 → switch1 → target2    ;    init3 → switch0 → switch2 → target4
init3 → switch0 → target5                    ;          init4 → switch0 → target1
init4 → switch0 → switch2 → target3    ;    init4 → switch0 → switch2 → target4
init4 → switch0 → target5
```

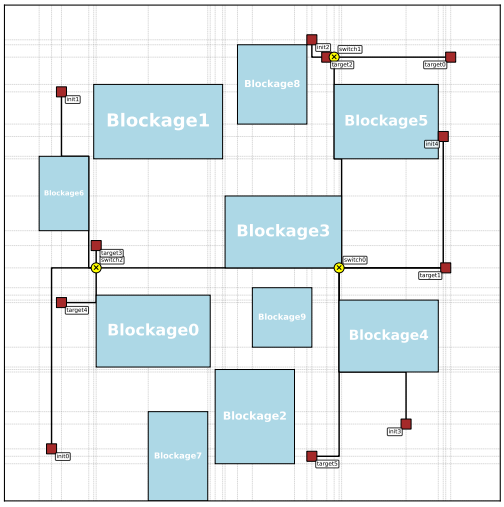
PPO, from scratch




```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.180, 0.470), (0.675, 0.470), (0.665, 0.895)

Active Route Paths:
init0 → switch0 → switch1 → switch2 → target0
init0 → switch0 → switch1 → target1          ;          init0 → switch0 → target3
init0 → switch0 → switch1 → target5                     ;
init1 → switch0 → switch1 → switch2 → target0
init1 → switch0 → switch1 → switch2 → target2    ;    init1 → switch0 → target4
init1 → switch0 → switch1 → target5    ;    init2 → switch2 → switch1 → target1
init2 → switch2 → target2    ;    init2 → switch2 → switch1 → switch0 → target3
init2 → switch2 → switch1 → target5    ;    init3 → switch1 → switch2 → target0
init3 → switch1 → switch2 → target2    ;    init3 → switch1 → switch0 → target4
init3 → switch1 → target5                    ;          init4 → switch1 → target1
init4 → switch1 → switch0 → target3    ;    init4 → switch1 → switch0 → target4
init4 → switch1 → target5
```

MCTS, pretrained

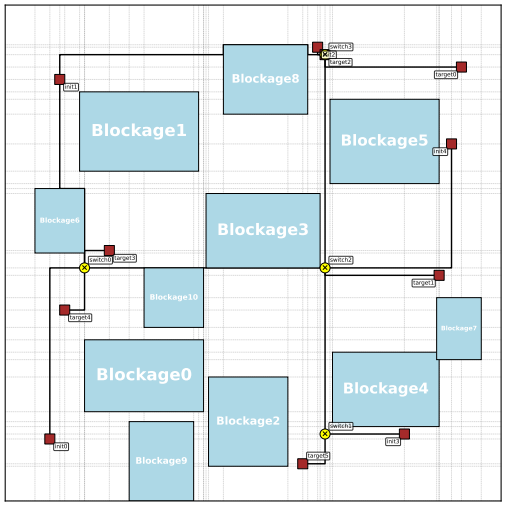



```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.675, 0.470), (0.665, 0.895), (0.185, 0.470)

Active Route Paths:
init0 → switch2 → switch0 → switch1 → target0
init0 → switch2 → switch0 → target1          ;          init0 → switch2 → target3
init0 → switch2 → switch0 → target5                     ;
init1 → switch2 → switch0 → switch1 → target0
init1 → switch2 → switch0 → switch1 → target2    ;    init1 → switch2 → target4
init1 → switch2 → switch0 → target5    ;    init2 → switch1 → switch0 → target1
init2 → switch1 → target2    ;    init2 → switch1 → switch0 → switch2 → target3
init2 → switch1 → switch0 → target5    ;    init3 → switch0 → switch1 → target0
init3 → switch0 → switch1 → target2    ;    init3 → switch0 → switch2 → target4
init3 → switch0 → target5                    ;          init4 → switch0 → target1
init4 → switch0 → switch2 → target3    ;    init4 → switch0 → switch2 → target4
init4 → switch0 → target5
```

MCTS, from scratch

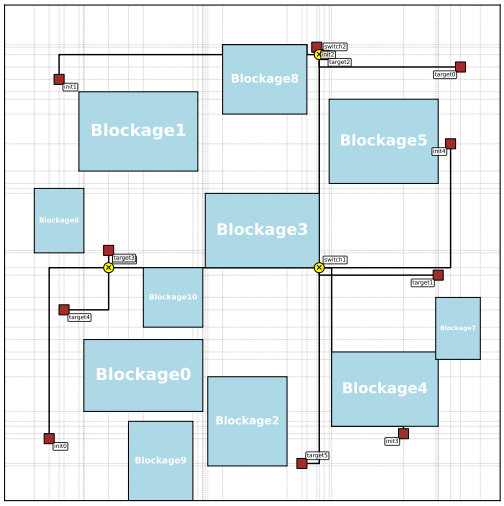
Figure 31: Fine-tuning instance 2.




```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.160, 0.470), (0.645, 0.135), (0.645, 0.470), (0.645, 0.900)

Active Route Paths:
init0 → switch0 → switch2 → switch3 → target0
init0 → switch0 → switch2 → target1          ;    init0 → switch0 → target3
init0 → switch0 → switch2 → switch1 → target5   ;    init1 → switch3 → target0
init1 → switch3 → target2                    ;    init1 → switch0 → target4
init1 → switch0 → switch2 → switch1 → target5
init2 → switch3 → switch2 → target1          ;    init2 → switch3 → target2
init2 → switch3 → switch2 → switch0 → target3
init2 → switch3 → switch2 → switch1 → target5
init3 → switch1 → switch2 → switch3 → target0
init3 → switch1 → switch2 → switch3 → target2
init3 → switch1 → switch2 → switch0 → target4   ;    init3 → switch1 → target5
init4 → switch2 → target1                    ;    init4 → switch2 → switch0 → target3
init4 → switch2 → switch0 → target4    ;    init4 → switch2 → switch1 → target5
```

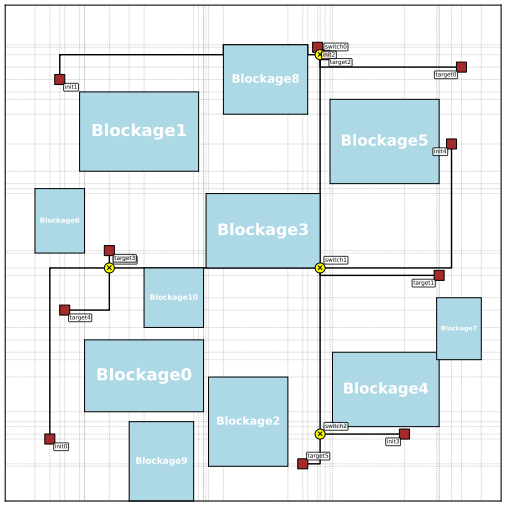
PPO, pretrained




```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.210, 0.470), (0.635, 0.470), (0.635, 0.900)

Active Route Paths:
init0 → switch0 → switch1 → switch2 → target0
init0 → switch0 → switch1 → target1          ;    init0 → switch0 → target3
init0 → switch0 → switch1 → target5          ;    init1 → switch2 → target0
init1 → switch2 → target2    ;    init1 → switch2 → switch1 → switch0 → target4
init1 → switch2 → switch1 → target5    ;    init2 → switch2 → switch1 → target1
init2 → switch2 → target2    ;    init2 → switch2 → switch1 → switch0 → target3
init2 → switch2 → switch1 → target5    ;    init3 → switch1 → switch2 → target0
init3 → switch1 → target2    ;    init3 → switch1 → switch0 → target4
init3 → switch1 → target5                    ;    init4 → switch1 → target1
init4 → switch1 → switch0 → target3    ;    init4 → switch1 → switch0 → target4
init4 → switch1 → target5
```

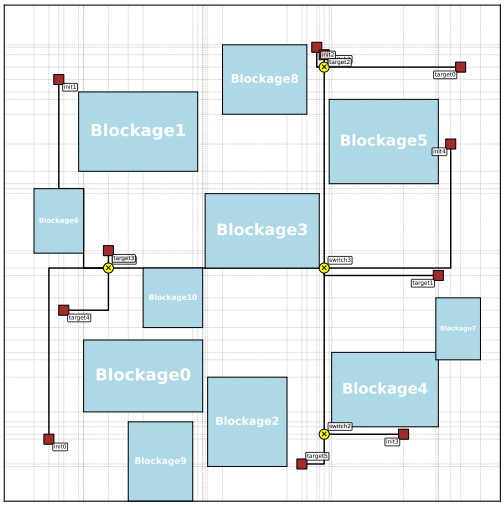
PPO, from scratch




```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.635, 0.900), (0.635, 0.470), (0.635, 0.135), (0.210, 0.470)

Active Route Paths:
init0 → switch3 → switch1 → switch0 → target0
init0 → switch3 → switch1 → target1          ;    init0 → switch3 → target3
init0 → switch3 → switch1 → switch2 → target5   ;    init1 → switch0 → target0
init1 → switch0 → target2    ;    init1 → switch0 → switch1 → switch3 → target4
init1 → switch0 → switch1 → switch2 → target5
init2 → switch0 → switch1 → target1          ;    init2 → switch0 → target2
init2 → switch0 → switch1 → switch3 → target3
init2 → switch0 → switch1 → switch2 → target5
init3 → switch2 → switch1 → switch0 → target0
init3 → switch2 → switch1 → switch0 → target2
init3 → switch2 → switch1 → switch3 → target4   ;    init3 → switch2 → target5
init4 → switch1 → target1                    ;    init4 → switch1 → switch3 → target3
init4 → switch1 → switch3 → target4    ;    init4 → switch1 → switch2 → target5
```

MCTS, pretrained

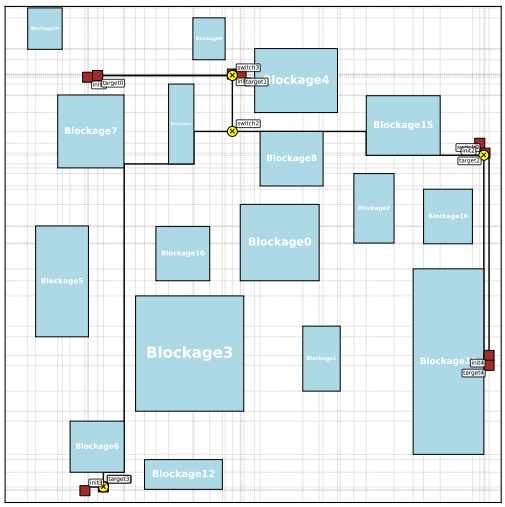



```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.210, 0.470), (0.645, 0.875), (0.645, 0.135), (0.645, 0.470)

Active Route Paths:
init0 → switch0 → switch3 → switch1 → target0
init0 → switch0 → switch3 → target1          ;    init0 → switch0 → target3
init0 → switch0 → switch3 → switch2 → target5
init1 → switch0 → switch3 → switch1 → target0
init1 → switch0 → switch3 → switch1 → target2    ;    init1 → switch0 → target4
init1 → switch0 → switch3 → switch2 → target5
init2 → switch1 → switch3 → target1          ;    init2 → switch1 → target2
init2 → switch1 → switch3 → switch0 → target3
init2 → switch1 → switch3 → switch2 → target5
init3 → switch2 → switch3 → switch1 → target0
init3 → switch2 → switch3 → switch1 → target2
init3 → switch2 → switch3 → switch0 → target4    ;    init3 → switch2 → target5
init4 → switch3 → target1          ;    init4 → switch3 → switch0 → target3
init4 → switch3 → switch0 → target4    ;    init4 → switch3 → switch2 → target5
```

MCTS, from scratch

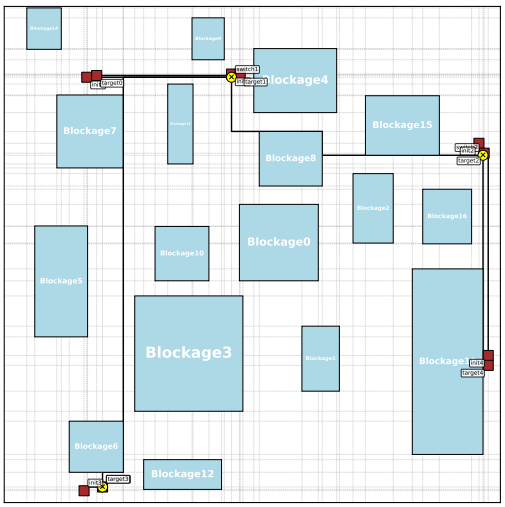
Figure 32: Fine-tuning instance 3.




```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.965, 0.700), (0.198, 0.031), (0.458, 0.748), (0.458, 0.861)

Active Route Paths:
init0 → switch3 → target0                    ;          init0 → switch3 → target1
init0 → switch3 → switch2 → switch0 → target2
init0 → switch3 → switch2 → switch1 → target3
init0 → switch3 → switch2 → switch0 → target4    ;    init1 → switch3 → target0
init1 → switch3 → target1    ;    init1 → switch3 → switch2 → switch0 → target2
init1 → switch3 → switch2 → switch1 → target3
init1 → switch3 → switch2 → switch0 → target4
init2 → switch0 → switch2 → switch3 → target0
init2 → switch0 → switch2 → switch3 → target1    ;    init2 → switch0 → target2
init2 → switch0 → switch2 → switch1 → target3    ;    init2 → switch0 → target4
init3 → switch1 → switch2 → switch3 → target0
init3 → switch1 → switch2 → switch3 → target1
init3 → switch1 → switch2 → switch0 → target2    ;    init3 → switch1 → target3
init3 → switch1 → switch2 → switch0 → target4
init4 → switch0 → switch2 → switch3 → target0
init4 → switch0 → switch2 → switch3 → target1    ;    init4 → switch0 → target2
init4 → switch0 → switch2 → switch1 → target3    ;    init4 → switch0 → target4
```

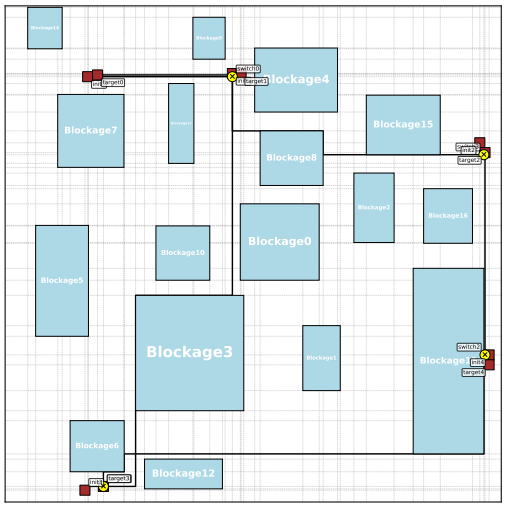
PPO, pretrained




```
NETLIST SUMMARY
========================
Total Switches: 3
Switch Positions:
(0.198, 0.031), (0.458, 0.857), (0.965, 0.700)

Active Route Paths:
init0 → switch1 → target0                    ;          init0 → switch1 → target1
init0 → switch1 → switch2 → target2    ;    init0 → switch1 → switch0 → target3
init0 → switch1 → switch2 → target4          ;          init1 → switch1 → target0
init1 → switch1 → target1                    ;          init1 → switch1 → switch2 → target2
init1 → switch1 → switch0 → target3    ;    init1 → switch1 → switch2 → target4
init2 → switch2 → switch1 → target0    ;    init2 → switch2 → switch1 → target1
init2 → switch2 → target2    ;    init2 → switch2 → switch1 → switch0 → target3
init2 → switch2 → target4          ;          init3 → switch0 → switch1 → target0
init3 → switch0 → switch1 → target1
init3 → switch0 → switch1 → switch2 → target2    ;    init3 → switch0 → target3
init3 → switch0 → switch1 → switch2 → target4
init4 → switch2 → switch1 → target0    ;    init4 → switch2 → switch1 → target1
init4 → switch2 → target2    ;    init4 → switch2 → switch1 → switch0 → target3
init4 → switch2 → target4
```

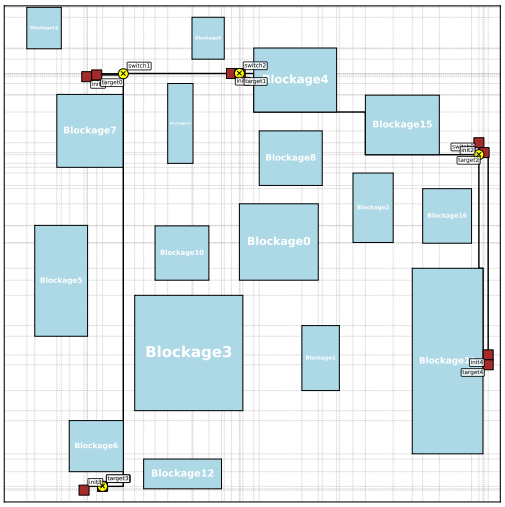
PPO, from scratch




```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.458, 0.857), (0.198, 0.031), (0.967, 0.297), (0.965, 0.700)

Active Route Paths:
init0 → switch0 → target0                    ;          init0 → switch0 → target1
init0 → switch0 → switch3 → target2    ;    init0 → switch0 → switch1 → target3
init0 → switch0 → switch3 → switch2 → target4    ;    init1 → switch0 → target0
init1 → switch0 → target1                    ;          init1 → switch0 → switch3 → target2
init1 → switch0 → switch1 → target3
init1 → switch0 → switch3 → switch2 → target4
init2 → switch3 → switch0 → target0    ;    init2 → switch3 → switch0 → target1
init2 → switch3 → target2    ;    init2 → switch3 → switch2 → switch1 → target3
init2 → switch3 → switch2 → target4    ;    init3 → switch1 → switch0 → target0
init3 → switch1 → switch0 → target1
init3 → switch1 → switch2 → switch3 → target2    ;    init3 → switch1 → target3
init3 → switch1 → switch2 → target4
init4 → switch2 → switch3 → switch0 → target0
init4 → switch2 → switch3 → switch0 → target1
init4 → switch2 → switch3 → target2    ;    init4 → switch2 → switch1 → target3
init4 → switch2 → target4
```

MCTS, pretrained




```
NETLIST SUMMARY
========================
Total Switches: 4
Switch Positions:
(0.957, 0.700), (0.240, 0.863), (0.474, 0.864), (0.198, 0.031)

Active Route Paths:
init0 → switch1 → target0                    ;          init0 → switch1 → switch2 → target1
init0 → switch1 → switch2 → switch0 → target2
init0 → switch1 → switch3 → target3
init0 → switch1 → switch2 → switch0 → target4
init1 → switch2 → switch1 → target0          ;          init1 → switch2 → target1
init1 → switch2 → switch0 → target2
init1 → switch2 → switch1 → switch3 → target3
init1 → switch2 → switch0 → target4
init2 → switch0 → switch2 → switch1 → target0
init2 → switch0 → switch2 → target1          ;          init2 → switch0 → target2
init2 → switch0 → switch2 → switch1 → switch3 → target3
init2 → switch0 → target4          ;          init3 → switch3 → switch1 → target0
init3 → switch3 → switch1 → switch2 → target1
init3 → switch3 → switch1 → switch2 → switch0 → target2
init3 → switch3 → switch2 → target3
init3 → switch3 → switch1 → switch2 → switch0 → target4
init4 → switch3 → switch0 → switch2 → target0
init4 → switch3 → switch0 → switch2 → switch1 → target0
init4 → switch0 → switch2 → target1          ;          init4 → switch0 → target2
init4 → switch0 → switch2 → switch1 → switch3 → target3
init4 → switch0 → target4
```

MCTS, from scratch

Figure 33: Fine-tuning instance 4.