Block-Level Goal Recognition Design: Technical Appendix
Tsz-Chiu Au

Abstract
This is the technical appendix for the paper entitled “Block-Level Goal Recognition Design” published in AAAI 2024. This document contains the pseudocodes, the definitions, and the examples that are omitted in the paper. Moreover, we present two new results for improving the performance of the GRD algorithms: correlated blocks and legal path bundles. Specifically, this document contains:
• The pseudocodes of the breadth-first search algorithm as described in the paper;
• The description and the pseudocodes of the local search algorithm;
• The definition of compact path trees;
• The definition and the analysis of correlated blocks, which states that if we can merge several correlated blocks into one block, the effect of combining correlated blocks is like pruning some branches in the search space of the complete GRD algorithms; and
• The definition and the analysis of legal path bundles, which is a technique for reducing the number of legal paths by implicitly representing a set of legal paths by a legal path bundle if certain assumptions hold.

Breadth-First Search with Pruned-Reduce and Design Subtree Pruning
Algorithm 1 is the pseudocode of the BFS with pruned-reduce and design subtree pruning. The inputs are the root block $b_{root}$, the set of legal paths $P_{leg}$, the pre-computed sets of blocks $B_{invalid}$ that possibly invalidate the legal paths, and the set of all blocks $B_{all}$. The BFS repeatedly takes the first design tree $\Theta$ from the queue and extends $\Theta$ with feasible blocks for its open regional vertices until the queue is empty. Pruned-reduce is implemented in Line 14, which only allows feasible blocks that possibly invalidate the legal paths in $P_{wcd}$ to be used for extending $\Theta$. Lines 10 and 12 implement a design subtree pruning rule. PrunedRegions($b_i$) is the set of regional vertices in $b_i$ that are pruned as described in the paper. Finally, the BFS returns a design tree with the minimum WCD.

Algorithm 1: BFS with Block-Level Pruned-Reduce and Design Subtree Pruning

Procedure BFSwithPruning($b_{root}, P_{leg}, B_{invalid}, B_{all}$)
1: $WCD_{min} := \infty$; $\theta_{min} := \emptyset$; $B_{invalid}^{wcd} := B_{all}$
2: Let $Q$ be a queue; add $\theta_{0} = \{ b_{root} \}$ to $Q$
3: while $Q$ is not empty do
4: Remove the first design tree $\Theta$ from $Q$
5: if $\Theta$ is encompassing and $WCD(\Theta) < WCD_{min}$ then
6: $WCD_{min} := WCD(\Theta)$; $\theta_{min} := \Theta$
7: $P_{wcd} := \{ \text{a subset of $P_{leg}$ s.t. } \text{prefix}(p_1, p_2) = \text{WCD}_{min} \text{ for all } p_1, p_2 \in P_{wcd} \}$
8: $B_{invalid}^{wcd} := \bigcup_{p \in P_{wcd}} B_{invalid}(p)$
9: for each non-terminal block $b_i \in \Theta$ do
10: Compute PrunedRegions($b_i$) if it does not exist.
11: for each open regional vertex $v_j \in V^r$ in $b_i$ do
12: if $v_j \notin \text{PrunedRegions}(b_i)$ then
13: for each subblock $b_k \in \text{dom}(v_j)$ do
14: if $b_k \in B_{invalid}^{wcd}$ then
15: Add $(\Theta \cup \{ b_k \})$ to the end of $Q$
16: return $\theta_{min}$

Local Search for Block-Level GRD
The BFS is inefficient since it is an optimal complete search algorithm. For large-scale GRD problems, we opt for a local search algorithm that can return a suboptimal design tree quickly. Algorithm 2 is the pseudocode of the algorithm. The algorithm uses the min-conflict heuristics that is highly effective for certain constrained optimization problems (Minton et al. 1992; Sosič and Gu 1994). The algorithm starts with a random design tree $\Theta$ that is full and encompassing. Then it iteratively improves it by randomly choosing a block in $\Theta$, replacing it with a random block, and generating a new random design subtree for it if there is none previously. Since the working principle of pruned-reduce suggests that we should focus on modifying the blocks that possibly invalidate the legal paths that yield the WCD, the algorithm prefers choosing such blocks in Line 17. We call this preference the pruned-reduce-like heuristic in the paper. Occasionally, it chooses a block randomly based on the $\epsilon$-greedy exploration strategy (Lines 14–17). Thus, the pruned-reduce-like heuristic only gives a higher priority to the blocks that possibly invalidate legal paths that yield the
Let \( b \) be \( \text{parent}[b'] \) be the parent of \( b' \) and \( v_r = v[b'] \) be the regional vertex in \( b \) that can be substituted by \( b' \). Let \( (V, V', E, E', V^\text{entry}, V^\text{exit}) \) be the specification of \( b' \), where \( G'' = (V, V', E, E') \) is the extended search space of \( b' \), \( V^\text{entry} \) is a set of entries, and \( V^\text{exit} \) is a set of exits. Let \( V^\text{in} \) and \( V^\text{out} \) be the incoming and outgoing vertices of \( v_r \) in \( b \), respectively.

Let \( p = (v_1, v_2, \ldots, v_m) \) be a subpath of a legal path that lies inside \( b \), where \( v_1 \) is an entry in \( b \) and \( v_m \) is an exit in \( b \). Note that \( p \) can go through \( G'' \) many times or do not go through \( G'' \) at all. Suppose \( p \) goes through \( G'' \) via the regional vertex \( v_r \). We want to identify a subpath \( p' \) of \( p \) that lies inside \( G' \) entirely.

If there is \((v_i, v_{i+1}) \in \text{edges}(p)\) s.t. \( v_j \in V^\text{in} \) and \( v_{j+1} \notin V \) for some \( 1 \leq i < m, v_{i+1} \) is the first vertex in the subpath \( p' \). If there is another edge \((v_j, v_{j+1}) \in \text{edges}(p)\) s.t. \( v_j \notin V \) and \( v_{j+1} \in (V \cap V^\text{out}) \) for some \( 1 \leq i < j < m, v_j \) is the last vertex in \( p' \). Then \( p' = \langle v_{i+1}, v_{i+2}, \ldots, v_j \rangle \).

We identify \( p' \), we create a compact path \( c \) by replacing \((v_{i+1}, v_{i+2}, \ldots, v_j)\) with the regional vertex \( v_j \) in \( p \). That is, \( c = (v_1, v_2, \ldots, v_i, v_j, v_{j+1}, v_{j+2}, \ldots, v_m) \).

However, suppose \((v_i, v_{i+1}) \) exists where \( v_i \in V^\text{in} \) and \( v_{i+1} \notin V \), but \((v_j, v_{j+1}) \) where \( v_j \notin V \) and \( v_{j+1} \) in \( V^\text{out} \) does not exist. It means that \( p \) entered \( G'' \) of \( b' \) and does not exit \( G' \) through \( V^\text{out} \). Since the goal vertices are ordinary vertices in the root block, \( p \) must have exited \( G' \) via another regional vertex \( v_{r1} \in V^\text{out} \), where \( v_{r1} \neq v_r \) and there is an edge in \( v_{r1} \) in the extended search space of \( b \). Then, we look into the exits of \( v_{r1} \) to see whether \( p \) exits from them. If not, we look into the exits of another regional vertex \( v_{r2} \) in the set of outgoing vertices of \( v_{r1} \), and so on. Eventually, \( p \) will exit from one of the regional vertices in \( b \). Let \((v_{r1}, v_{r2}, \ldots, v_{rt}) \) be the chain of regional vertices \( p \) goes through. Although we do not know which section of the subpath \( p \) belongs to \( G'' \) and which section belongs to the feasible blocks of other regional vertices in the chain, we create a compact path \( c = (v_{r1}, v_{r2}, \ldots, v_{rt}, v_{r1+1}, v_{r2+1}, \ldots, v_{mr}) \).

We need to check whether the remaining of \( c \) (i.e., \( (v_{r1}, v_{r2}, \ldots, v_{mr}) \)) passes through the other regional vertices in \( b \) or re-enters some of the regional vertices in \( (v_r, v_{r1}, \ldots, v_{rj}) \). If so, we replace the subpaths in \( c \) with the corresponding regional vertices. In the end, we get a new compact path for \( p \) that includes all regional vertices \( p \) goes through.

Since a regional vertex \( v \) in \( c \) can appear multiple times in \( c \) since \( p \) can go through \( v \) multiple times. To distinguish all instances of a regional vertex \( v \) in \( c \), we label them differently: we add a superscript \( i \) to \( v \) for the \( i \)th instance of \( v \) in \( c \), such that all regional vertices in \( c \) are different (i.e., \( v^1, v^2, \ldots, v^k \) are the different regional vertices in \( c \) but the same regional vertex in \( p \)).

A compact path tree \( T^\text{compact} \) for a given set \( P \) of subpaths is a tree with both ordinary vertices and regional vertices, and each path from the root of \( T^\text{compact} \) to a terminal vertex of \( T^\text{compact} \) is a compact path of a subpath \( p \in P \). Figure 1 shows the compact path tree of the example in the paper. If two different subpaths \( p_1, p_2 \in P \)'s junction is \( p \) (e.g., \( C3 \) in Figure 1), they will share the same trunks in \( T^\text{compact} \) for the vertices in \( \text{prefix}(p_1, p_2) \). Moreover, for each junction with \( n \) branches in the legal path tree \( T \) formed by \( P \), there is a junction with \( n \) branches in \( T^\text{compact} \).

We construct a compact path tree \( T^\text{compact} \) from a legal path tree \( T \) as follows. Initially, \( T^\text{compact} \) is a legal path tree formed by combining the paths in \( T \). For each subpath \( p \in P \), if a subpath \( p' \) of \( p \) is substituted by a regional vertex \( v' \) in the corresponding compact path, \( p' \) in \( T^\text{compact} \) will be substituted by \( v' \) as well. Moreover, if a subpath \( p'' \) of \( p \) is substituted by a sequence of regional vertices \( (v_1, v_2, \ldots, v_m) \) in the corresponding compact path, \( p'' \)
Figure 1: A legal path tree $T$ for the example in Figure 1 in the paper. The tree on the right is the corresponding compact path tree $T^{\text{compact}}$, which is the same as $T$ except that the set of states inside the regional vertex $v_4$ in $T$ are replaced by $v_6$.

in $T^{\text{compact}}$ will be substituted by $\langle v'_1, v'_2, \ldots, v'_m \rangle$.

However, if $p'$ goes through a junction in $T$, we must add additional branches to $v'$ or some regional vertices in $\langle v'_1, v'_2, \ldots, v'_m \rangle$. When $p'$ is substituted by $v'$ only, we must create different branches after $v'$ for different branches of $v$ in $T$. In other words, if $v$ is a junction in $T$ with two or more child vertices and $p'$ goes through $v$ to one of the child vertices while $p'$ has entered a regional vertex $v'$, the subpaths that go to other child vertices of $v$ also enter $v'$. Thus, these subpaths will be substituted by $v'$ or a sequence of regional vertices starting with $v'$ in $T^{\text{compact}}$. Therefore, we must add a new branch after $v'$ in $T^{\text{compact}}$ for each branch of $v$ in $T$. When $p'$ is substituted by $\langle v'_1, v'_2, \ldots, v'_m \rangle$, we must add additional branches to $v'_i$ for some $1 \leq i \leq m$ to $T^{\text{compact}}$ for other subpath $p''$ that goes through $v$ in $T$ but $p'' \neq p'$. Here, $v'_i$ is the regional vertex from which $p''$ leaves the sequence $\langle v'_1, v'_2, \ldots, v'_m \rangle$.

**An Example of Design Subtree Pruning**

This section provides an example to illustrate how the design subtree pruning rule works. Figure 2 shows an environment with eight legal paths to different goals. The root block $b_{\text{root}}$ has three regional vertices: $v_1$, $v_2$, and $v_3$. The domains of $v_1$ and $v_3$ are $\text{dom}(v_1) = \{b_1, b_2\}$ and $\text{dom}(v_3) = \{b_5, b_6\}$, respectively. These blocks act like on-off switches for the legal paths to $g_2$ and $g_7$. The domain of $v_2$ is $\text{dom}(v_2) = \{b_3, b_4\}$. This pair of blocks acts like a toggle between $g_4$ and $g_5$ such that the environment design can only choose to allow one legal path to reach either $g_4$ or $g_5$ but not both. In short, $\text{dom}(v_2)$ enforces a design constraint that $g_4$ and $g_5$ are mutually exclusive.

We shall follow the definitions in Section “Pruning Design Subtrees” in the paper to define the legal path trees in this example. Let $P = P^{\text{legal}}$. The middle of Figure 2 shows the legal path tree $T$ formed by combining the legal paths in $P$. Then, we construct a compact path tree $T^{\text{compact}}$ by replacing the subpaths in $T$ that go through the regions with the regional vertices, as described in the previous section. More precisely, (B9) is replaced by $v_1$, (D9) is replaced by $v_2$, (E9) is replaced by $v_2$, and (G9) is replaced by $v_3$. The tree at the right side of Figure 2 is $T^{\text{compact}}$.

Figure 3 shows how to calculate the lower and upper bounds of the relative WCDs of the vertices in the legal path tree $T$ according to the instructions in the paper. Since the environment is a root block $b_{\text{root}}$, the vertices of the initial state $s_0$ and the goal states are ordinary vertices. In addition, we set the lower and upper bounds of these terminal vertices (i.e., the goal vertices) in $T$ as zero. We shall use $T$ and $T^{\text{min}}$ to compute the lower and upper bounds of the internal vertices in $T$. $T^{\text{min}}$ is derived from $T$ by removing all the legal paths in $T$ that any feasible block of any regional vertex could possibly invalidate. In this example, there are six feasible blocks, and they invalidate four legal paths: $b_2$ invalidates the legal path to $g_2$; $b_3$ invalidates the legal path to $g_5$; $b_4$ invalidates the legal path to $g_4$; and $b_6$ invalidates the legal path to $g_7$. Hence, we remove these legal paths from $T$ to form $T^{\text{min}}$ as shown in the tree in the middle of Figure 3.

Since the remaining four paths in $T^{\text{min}}$ are not invalidated by any blocks, they will never be invalidated regardless of the choice of the design subtrees of the regional vertices. Therefore, the relative WCDs of the internal vertices in $T^{\text{min}}$, computed by Equation 3, are the lower bounds of the relative WCDs of the internal vertices. It is because no matter which design subtrees are chosen, the set of valid legal paths must include those in $T^{\text{min}}$. Adding more legal paths to $T^{\text{min}}$ could only increase the relative WCDs. Hence, these relative WCDs are the lower bounds of the relative WDs after the design subtrees are chosen.

For the sake of completeness, for every vertex $v$ in $T$ but not in $T^{\text{min}}$, we set the lower bound of the relative WCD of $v$ to 0. It is because the legal paths going through $v$ may or may not be invalidated eventually. If they are not invalidated, $v$ will have a relative WCD. In the worst case, only one legal path is going through $v$ that is not invalidated such that the
Figure 2: An environment with three regional vertices: $v_1$, $v_2$, and $v_3$, where $\text{dom}(v_1) = \{b_1, b_2\}$, $\text{dom}(v_2) = \{b_3, b_4\}$, and $\text{dom}(v_3) = \{b_5, b_6\}$. There are eight legal paths. The legal path tree $T$ and the compact path tree $T^{\text{compact}}$ of the legal paths are shown in the figure.

Figure 3: In the legal path tree $T$, the number adjacent to a vertex is the upper bound of the relative WCD of the state. In the legal path tree $T^{\text{min}}$, the number adjacent to a vertex is the lower bound of the relative WCD of the state. In $T^{\text{compact}}$, the pair of numbers adjacent to C2 and E3 are the lower and upper bounds of the relative WCDs of the state. The two crossed vertices in $T^{\text{compact}}$ are marked as “pruned”. However, only $v_1$ is put in $\text{PrunedRegions}(b_{\text{root}})$, where $b_{\text{root}}$ is the root block of this environment.
relative WCD of $v$ is 0. Thus, the lower bound of the relative WCD of $v$ is 0.

Similarly, the relative WCDs of the internal vertices in $T$, computed by Equation 3 in the paper, are the upper bounds of the relative WCDs of the internal vertices. It is because no matter which design subtrees are chosen, the set of valid legal paths must be a subset of $T^{\min}$. Removing some legal paths from $T$ could only decrease the relative WCDs. Hence, these relative WCDs are the upper bounds of the relative WCDs after design subtrees are selected.

After calculating the lower and upper bounds of the relative WCDs of the internal vertices in $T$, we check every junction $v$ in $T$ to see whether there are two child vertices $v_1, v_2 \in \text{children}(v)$ of $v$ such that the lower bound of the relative WCD of $v_1$ is greater than the upper bound of the relative WCD of $v_2$. In Figure 3, $T$ has seven junctions (D2, C3, B4, C4, F6, F7, G8). The only junction that satisfies this condition is D2, because D2 has two child vertices C2 and E2, and the lower bound of the relative WCD of E2 is 6 and the upper bound of the relative WCD of C2 is 4. Since the upper bound of the relative WCD of C2 is smaller than the lower bound of the relative WCD of E2, we mark all regional vertices in the subtree of C2 in $T^{\text{compact}}$ as “pruned”. The subtree of C2 in $T^{\text{compact}}$ has two regional vertices: $v_1$ and $v_2$, and both are marked as “pruned” as shown in the right subfigure in Figure 3.

However, only $v_1$ is put in $\text{PrunedRegions}(b_{\text{root}})$ for the BFS to prune the design subtrees of $v_1$. We do not put $v_2$ in $\text{PrunedRegions}(b_{\text{root}})$ because $v_2$ occurs twice in $T^{\text{compact}}$ and not all instances of $v_2$ are marked as “pruned” (more specifically, $v_2$ before E10 is not marked as “pruned”). Thus, the BFS still needs to expand the open regional vertex $v_2$ in the design tree since this affects the WCD of the design tree via the legal path to $v_2$. By contrast, the minimum WCD would not be affected by the choice of the design subtree of $v_1$—the relative WCD of D2 remains the same regardless of the block used for substituting $v_1$. Since the relative WCD of D2 is independent of the design subtree of $v_1$, the minimum WCD, which is the relative WCD of D1 minus 1, is also independent of the design subtree of $v_1$. Therefore, the GRD algorithms can avoid the expansion of $v_2$ to save time. The pseudocodes of the BFS and the local search algorithm in this document have already utilized $\text{PrunedRegions}(b_{\text{root}})$ to prune $v_2$.

The pruning rule works in this simple environment even if the blocks for the regional vertices are more complicated and their child blocks contain subblocks. In this case, when we compute $T^{\min}$, we have to use all possible subblocks to check for the invalidation of legal paths in $T$.

If the environment is not a root block but a child block $b$ of another block parent[$b$] such that $s_0$ is not an initial vertex but an entry of $b$ and the goal vertices are the exits of $b$, we cannot assume the lower and upper bounds of the relative WCDs of the exits are zero. Instead, the lower and upper bounds of the relative WCDs of the exits are computed from their child vertices in the legal paths. These child vertices are not in $b$. If they are ordinary vertices in parent[$b$], we can obtain the lower and upper bounds of the relative WCDs of the child vertices in parent[$b$]. However, this requires us to compute the lower and upper bounds of the relative WCDs of the vertices in parent[$b$] before compute the lower and upper bounds of the relative WCDs of the vertices in $b$. Fortunately, if we apply the design subtree pruning rule of the blocks in the hierarchical design space in a top-down manner, parent[$b$] is always evaluated before $b$. If some of the child vertices are not ordinary vertices in parent[$b$], they must be ordinary vertices in some feasible blocks of another regional vertex $v'$ in parent[$b$], and $v'$ is adjacent to the regional vertex of $b$ in parent[$b$]. In this case, we need to evaluate the chain of regional vertices in parent[$b$] together as if one regional vertex.

The example in Figure 3 highlights the situations in which pruning design subtrees can likely occur. In Figure 3, we can see that the junctions in the left subtree of D2 in $T$ are much closer to $s_0$ than the junctions in the right subtree of D2. Since the junctions are the vertices that potentially yield the WCD, we can deduce that the relative WCDs of the vertices in the right subtree must be larger than that in the left subtree. Thus, it is likely that the vertices that yield the minimum WCDs are in the right subtree for any design tree. The design subtree pruning looks for the discrepancy of the locations of the junctions in the subtrees and ignores the regional vertices in the subtree whose junctions are close to the root of the legal path tree.

The effectiveness of this pruning rule depends on other factors as well. For example, if the pruned regional vertices have some large design subtrees, our GRD algorithms can speed up tremendously by avoiding these design subtrees.

**Performance Improvement by Correlated Blocks**

The performance gain of our block-level GRD algorithms stems from using blocks, which merge several correlated modifications into one modification. We can apply the same idea by merging several correlated blocks into one. For example, in Figure 4, there are two regional vertices $v_1$ and $v_2$, and $\text{dom}(v_1) = \text{dom}(v_2) = \{b_1, b_2, b_3\}$. If a design constraint is that the region represented by $v_1$ and $v_2$ must connect to each other all the time, we cannot substitute $b_2$ for $v_1$ or $b_1$ for $v_2$. In general, if only some combinations of the blocks in $\text{dom}(v_1)$ and $\text{dom}(v_2)$ of two adjacent regional vertices $v_1$ and $v_2$ are allowed, we can merge $v_1$ and $v_2$ into one regional vertex $v_3$ such that $\text{dom}(v_3) \subseteq \text{dom}(v_1) \times \text{dom}(v_2)$. For example, in Figure 4, we have $\text{dom}(v_3) = \{(b_1, b_2), (b_1, b_3), (b_3, b_2), (b_3, b_3)\}$. Since $\text{dom}(v_3)$ is much less than $\text{dom}(v_1) \times \text{dom}(v_2)$, the performance speedup is achieved by avoiding the enumeration of all possible combinations of $\text{dom}(v_1)$ and $\text{dom}(v_2)$.

Suppose 1) every block has $k$ regional vertices, 2) the domain size of every regional vertex is $m$, and 3) all design trees are full and have a height of $h$. There are $\frac{m^{k(k+1)-1}}{m^k-1}$ different design trees. Now suppose 1) the set of $k$ regional vertices can be partitioned into $k/g$ groups such that the child blocks of the $g$ regional vertices in a group are correlated, where $1 < g \leq k/2$, and 2) the size of the combined domain $\prod_{v_i \in V^r} \text{dom}(v_i)$ is $\alpha \prod_{v_i \in V^r} |\text{dom}(v_i)|$, where $V^r$ is a group of regional vertices whose child blocks
are correlated and $0 < \alpha \leq 1$. After combining the correlated blocks, the number of different design trees is roughly \((\alpha m)^{k(b+1)}-1 \). If \(\alpha = 1\), the number of design trees remains the same. If \(\alpha < 1\), the “branching factor” \(m\) is multiplied by \(\alpha\), which is less than 1. Then, the effect of combining correlated blocks is like pruning some branches in the search space of the complete GRD algorithms.

Reducing the Number of Legal Paths by Legal Path Bundles

The existing GRD works assume that the set of legal paths is given beforehand or can be generated quickly. Typically, the set \(P_{leg}\) of legal paths is small, so agents can only follow some chosen paths (e.g., the shortest paths) to reach their goals. However, if agents are truly allowed to move freely in an environment, the set of feasible paths agents can choose can be huge. In the worst case, the number of feasible paths in a search space is exponential to the size of the search space. The computation of the WCD would become too time-consuming if \(P_{leg}\) is too large. Moreover, a computer could run out of memory if we explicitly store all feasible paths in \(P_{leg}\).

This section describes an approach to handle many feasible paths in \(P_{leg}\). This approach depends on some assumptions we make for the paths and blocks, such that we can implicitly represent a set of related legal paths by a legal path bundle. At the same time, the computation of the WCDs remains the same with legal path bundles. For example, in Figure 5, a subset of legal paths \(P \subseteq P_{leg}\) can be replaced by just one legal path in \(P_{bundle}\) without changing the WCD if certain conditions are satisfied. In this section, we state a sufficient condition for a successful replacement of \(P\) with \(P_{bundle}\) and prove that the WCD remains unchanged under the condition.

Motivating Example

First, we consider a simple case in which there is one regional vertex \(v\) in the root block \(b_{root}\), and \(v\) has one child block \(b\), one incoming vertex in \(V_{in}\), and one outgoing vertex in \(V_{out}\). Suppose all legal paths in \(P_{leg}\) have to go through \(b\). For every path \(p_i \in P_{leg}\), we partition \(p_i\) into three subpaths: 1) \(p_i^{prefix}\) is the subpath of \(p_i\) that lies inside \(b\) entirely, 2) \(p_i^{suffix}\) is the prefix of \(p_i\) before \(p_i^{prefix}\), and 3) \(p_i^{suffix}\) is the suffix of \(p_i\) after \(p_i^{prefix}\). Let \(p_i^b = p_i^{prefix} = p_i^{suffix}\). We say \(p_i\) is the complete path of \(p_i^{prefix}\) and \(p_i^{suffix}\), and \(p_i^{prefix}\). The collection of these subpaths are stored in \(P_{leg}^b = \{p_i^b\}_{p_i \in P_{leg}}\), \(P_{leg}^{prefix} = \{p_i^{prefix}\}_{p_i \in P_{leg}}\), and \(P_{leg}^{suffix} = \{p_i^{suffix}\}_{p_i \in P_{leg}}\).

Let \(G = \{g_1, g_2, \ldots, g_n\}\) be the set of all goals. We further partition \(P_{leg}^b\) into \(P_{leg}^{goal} = \{p_i^b\}_{p_i \in P_{leg}^{goal}}\) such that for every \(p_i^b \in P_{leg}^{goal}\), the goal of the complete path \(p_i\) of \(p_i^b\) is \(g[p_i] = g_j\), for \(1 \leq j \leq n\).

The first assumption for forming legal path bundles is:

**Definition 1** \(P_{leg}^b\) is goal-equivalent if and only if \(P_{leg}^{goal}\) is goal-equivalent for all \(P_{leg}^b\), where \(1 \leq i \leq j \leq n\).

This definition states that the set of subpaths for every goal in \(P_{leg}^{goal}\) are the same in a goal-equivalent \(P_{leg}^b\). For any two different goals \(g_1\) and \(g_2\), if there is a subpath \(p_i^b \in P_{leg}^{goal}\) s.t. path \(p_i^{prefix}\) reaches \(g_j\), there is also another subpath \(p_k^b \in P_{leg}^{goal}\) s.t. path \(p_k^{prefix}\) reaches \(g_j\) and \(p_k^b = p_i^b\).

Likewise, we partition \(P_{leg}^{suffix}\) into \(P_{leg}^{suffix} = \{p_i^{suffix}\}_{p_i \in P_{leg}^{suffix}}\) for all \(p_i^b \in P_{leg}^b\), where \(P_{leg}^{suffix}\) is the set of all suffixes whose complete paths contain \(p_i^b\). The second assumption for forming legal path bundles is:

**Definition 2** \(P_{leg}^{suffix}\) is suffix-equivalent if and only if \(P_{leg}^{suffix}\) is suffix-equivalent for any \(P_{leg}^b\) s.t. \(p_i^b = p_j^b\).

In other words, if two subpaths in \(P_{leg}^b\) are the same, the set of suffixes of the corresponding complete paths are also the same.

The goal-equivalence of \(P_{leg}^b\) and the suffix-equivalence of \(P_{leg}^{suffix}\) allow us to replace a subpath in \(P_{leg}^b\) with another subpath in \(P_{leg}^b\) for the same goal and this would not affect the subpaths in \(P_{leg}^{suffix}\) since their suffixes and their goals are the same.

Let \(p_{longest}^b \in P_{leg}^b\) be the longest subpath in \(P_{leg}^b\). For each legal path \(p_i \in P_{leg}^b\), we replace the subpath \(p_i^b\) in \(p_i\) by \(p_{longest}^b\) to form a new legal path \(p_i^{bundle}\). Let \(P_{bundle}\) be the set of legal paths after replacing \(p_i^b\) with \(p_{longest}^b\). Obviously, \(P_{leg}^{bundle}\) is a subset of \(P_{leg}^b\) since it only includes the legal paths whose subpaths in \(P_{leg}^b\) are \(p_{longest}^b\). The following theorem states that if we substitute \(P_{leg}^{bundle}\) for \(P_{leg}^b\), the WCD remains the same.
Theorem 1 If 1) \( p_1^{\text{prefix}} = p_2^{\text{prefix}} \) for all \( p_i^{\text{prefix}}, p_j^{\text{prefix}} \in P^\text{leg}_b \); 2) \( P^\text{leg}_b \) is goal-equivalent, and 3) \( P^\text{leg}_b \) is suffix-equivalent, the WCD of \( P^\text{leg} \) and the WCD of \( P^\text{leg}_b \) bundle are the same.

Proof First, since all subpaths in \( p_i^{\text{prefix}} \) are the same, the WCD of \( P^\text{leg} \) must be larger than the length of any subpath in \( p_i^{\text{prefix}} \). Second, we prove by contradiction that the state that yields the WCD does not exist in the subpaths in \( P^\text{leg}_b \).

Assume the state that yields the WCD is the last state of prefix \( (p_i^{\text{prefix}}, p_2^{\text{prefix}}) \), where \( p_i^{\text{prefix}}, p_2^{\text{prefix}} \in P^\text{leg}_b \) such that the goals \( g_1, g_2 \) of their corresponding complete paths are different, respectively, but \( p_i^{\text{prefix}} \neq p_2^{\text{prefix}} \). Without loss of generality, let \( |p_i^{\text{prefix}}| > |p_2^{\text{prefix}}| \). We have \( |\text{prefix}(p_i^{\text{prefix}}, p_2^{\text{prefix}})| < |p_i^{\text{prefix}}| \) since \( p_i^{\text{prefix}} \) and \( p_2^{\text{prefix}} \) are different. Since \( P^\text{leg}_b \) is goal-equivalent, there exists a subpath \( p_b \in P^\text{leg}_b \) that is the same as \( p_i^{\text{prefix}} \) but the goal of the corresponding complete path is \( g_2 \) instead of \( g_1 \). Clearly, the state that yields the WCD is not the last state of prefix \( (p_b^{\text{prefix}}, p_2^{\text{prefix}}) \) since \( |\text{prefix}(p_b^{\text{prefix}}, p_2^{\text{prefix}})| = |p_i^{\text{prefix}}| > |p_b^{\text{prefix}}| \). Hence, by contradiction, there are no \( p_i^{\text{prefix}}, p_2^{\text{prefix}} \in P^\text{leg}_b \) such that the last state of prefix \( (p_i^{\text{prefix}}, p_2^{\text{prefix}}) \) yields the WCD. Third, let \( s \) be some path in \( P^\text{leg}_b \) be the state that yields the WCD. Since \( P^\text{leg}_b \) is suffix-equivalent, the relative WCD of the first state of any subpath in \( P^\text{leg}_b \) are the same. Since all subpaths in \( p_i^{\text{prefix}} \) have the same length, there exist complete legal paths \( p_1, p_2 \in P^\text{leg} \) where the last state of their common prefix is \( s \) such that their subpaths in \( P^\text{leg}_b \) are \( p_b^{\text{leg}_b} \). It turns out both \( p_1 \) and \( p_2 \) are in \( P^\text{leg}_b \) bundle, and hence the WCD of \( P^\text{leg}_b \) bundle are the same as the WCD yields \( s \) in \( P^\text{leg} \).

Since the WCD of \( P^\text{leg}_b \) bundle is equal to the WCD of \( P^\text{leg} \) but \( P^\text{leg}_b \) bundle is much smaller than \( P^\text{leg} \), we could substitute \( P^\text{leg}_b \) bundle for \( P^\text{leg} \) in our GRD algorithms and yet the algorithms produce the same design tree with the minimum WCD. Hence, even if we include all kinds of variants of any subpath in \( P^\text{leg}_b \) so that \( P^\text{leg} \) becomes large, the size of \( P^\text{leg}_b \) bundle remains the same, and we can use \( P^\text{leg}_b \) bundle to compute the optimal design tree with the same minimum WCD.

Goal-Equivalence and RWCD-equivalence

We can generalize Theorem 1 by rewriting the theorem in terms of the relative WCD of the subpaths as follows.

Definition 3 Given 1) two ordinary vertices \( v_1, v_2 \in V \) in an extended search space \( G = (V, V^r, E, E^r) \) and 2) a subset \( P \subseteq P^\text{leg} \) of all legal paths such that a) all paths in \( P \) first go through \( v_1 \) and then go through \( v_2 \) (more precisely, if subpath \( (p, v_1, v_2) \) is the subpath of \( p \) from \( v_1 \) to \( v_2 \), inclusively, then \( v \in V \) for all \( v \in \) subpath \( (v_1, v_2) \) in \( E \) for all edge \( (v, v') \) on subpath \( (v_1, v_2) \) and b) the prefix before \( v_1 \) of any path in \( P \) are the same, we say \( P \) is goal-equivalent for \( v_1 \) and \( v_2 \) if and only if the set of all subpaths \( \{\text{subpath}(p, v_1, v_2)\} \) in \( P \) can be partitioned into \( P = \{G_i, i \in G\} \) such that \( P^\text{leg}_i = P^\text{leg}_j \) for any \( G_i, G_j \in G \) where \( G = \{g[p] \} \) and \( |G| \geq 2 \).

Note that \( |G'| \) has to be larger than or equal to 2 in Definition 3.

Definition 4 Given 1) two ordinary vertices \( v_1, v_2 \in V \) and 2) a subset \( P \subseteq P^\text{leg} \) of all legal paths in \( P^\text{leg} \) that first go through \( v_1 \) and then go through \( v_2 \) and has the same prefix before \( v_1 \), we say \( P \) is RWCD-equivalent for \( v_1 \) and \( v_2 \) if and only all relative WCDs \( \text{RWCD}(\Theta, v) \) of all \( v \) in the legal path tree formed by combining the prefixes of the paths in \( T \) are the same for any design tree \( \Theta \).

Suffix-equivalence in Definition 2 requires that the subtrees at \( v \) in the legal path tree are the same. Hence, suffix-equivalence will satisfy the conditions in Definition 4.

Theorem 2 Given 1) two ordinary vertices \( v_1, v_2 \in V \) and 2) a subset \( P \subseteq P^\text{leg} \) of all legal paths in \( P^\text{leg} \) that first go through \( v_1 \) and then go through \( v_2 \), the relative WCD at \( v_1 \) in the legal path tree formed by \( P \) is the same as the relative WCD at \( v_1 \) in the legal path tree formed by \( P^\text{leg}_b \) for any design tree \( \Theta \), where \( P^\text{leg}_b \) is a subset of paths in \( P \) whose subpath \( (p, v_1, v_2) = \text{subpath}(p|\text{longest}, v_1, v_2) \) where \( \text{longest} = \arg \max_{p \in P} \{\text{subpath}(p, v_1, v_2)\} \) is the path with the longest subpath between \( v_1 \) and \( v_2 \).

Proof Let us assume RWCD1 at \( v_1 \) in the legal path tree formed by \( P \) is different from RWCD2 at \( v_1 \) in the legal path tree formed by \( P^\text{leg}_b \). It means that there are two legal paths \( p_1, p_2 \in P \) that yields RWCD1 at \( v_1 \), but \( p_1 \neq p_2 \). Due to goal-equivalence, there exists \( p_3 \in P \) such that \( p_3 = p_1|\text{longest} \) and \( g[p_3] = g[p_1] \). Likewise, there exists \( p_4 \in P \) such that \( p_4 = p_2|\text{longest} \) and \( g[p_4] = g[p_2] \). Then RWCD3 at \( v_1 \) formed by \( p_1 \) and \( p_2 \) will be larger than RWCD1, which contradicts the fact that RWCD1 is a relative WCD at \( v_1 \). Hence, we have RWCD1 = RWCD2.

We can apply Theorem 2 to replace a subset of legal paths \( P \subseteq P^\text{leg} \) with \( P^\text{leg}_b \) such that \( |P^\text{leg}_b| \leq |P| \) while keeping the WCD unchanged. Although the conditions for goal-equivalence and RWCD-equivalence are quite restrictive, they often hold in “passage” blocks in which all agents simply go from entries to exits regardless of their goals. The goal-equivalence permits minor deviations from the optimal paths in these passage blocks.

Conclusions

This technical appendix provided the missing information in our paper entitled “Block-Level Goal Recognition Design” published in AAAI 2024. These missing information are the pseudocodes of the algorithms and the definition of compact path trees. We also provide an example showing how the design subtree pruning rule works. Moreover, we presented two new techniques to improve the block-level GRD algorithms: collected blocks and legal path bundles. The former merges several correlated blocks into one block to reduce the search space of the GRD algorithms. The latter reduces the number of legal paths by implicitly representing a set of legal paths by a legal path bundle. Both ideas utilize the properties of blocks to speed up the GRD algorithms. Blocks could offer other properties that can be helpful to simplify
the GRD problems. In the future, we would like to discover these properties for solving large-scale GRD problems.

Acknowledgments
This work has been taken place at UNIST and was supported by NRF (2022R1A2C101216812).

References