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;
- An example showing how the design subtree pruning rule works;
- 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 Θ from the queue and extends Θ 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 Θ. Lines 10 and 12 implement a design subtree pruning rule. $\mathsf{PrunedRegions}(b_i)$ 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($b_{\text{root}}, P^{leg}, B^{\text{invalid}}, B_{\text{all}}$)
 1: $\text{WCD}_{\text{min}} := \infty$; $\Theta_{\text{min}} := \emptyset$; $B^{\text{invalid}}_{\text{wcd}} := B_{\text{all}}$
 2: Let $Q$ be a queue; add $\Theta_0 = \{b_{\text{root}}\}$ to $Q$
 3: while $Q$ is not empty do
 4:   Remove the first design tree $\Theta$ from $Q$
 5:   if $\Theta$ is encompassing and $\text{WCD}(\Theta) < \text{WCD}_{\text{min}}$ then
 6:     $\text{WCD}_{\text{min}} := \text{WCD}(\Theta)$; $\Theta_{\text{min}} := \Theta$
 7:     $P_{\text{wcd}}$ := a subset of $P^{leg}$ s.t. $|\mathsf{prefix}(p_1, p_2)| =$
            $\text{WCD}_{\text{min}}$ for all $p_1, p_2 \in P_{\text{wcd}}$
 8:     $B^{\text{invalid}}_{\text{wcd}} := \bigcup_{p \in P_{\text{wcd}}} B^{\text{invalid}}(p)$
 9:   for each non-terminal block $b_i \in \Theta$ do
10:     Compute $\mathsf{PrunedRegions}(b_i)$ if it does not exist.
11:     for each open regional vertex $v_j \in V^r$ in $b_i$ do
12:       if $v_j \notin \mathsf{PrunedRegions}(b_i)$ then
13:         for each subblock $b_k \in \mathsf{dom}(v_j)$ do
14:           if $b_k \in B^{\text{invalid}}_{\text{wcd}}$ then
15:             Add $(\Theta \cup \{b_k\})$ to the end of $Q$
16: return $\Theta_{\text{min}}$
```

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 Θ that is full and encompassing. Then it iteratively improves it by randomly choosing a block in Θ, 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 ϵ-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

Algorithm 2: Local Search for Block-Level GRD

```
Procedure LocalSearch($b_{\mathsf{root}}, P^{leg}, B^{\mathsf{invalid}}, B_{\mathsf{all}}$)
1:  WCD$_{\mathsf{min}} := \infty$; $\Theta_{\mathsf{min}} := \emptyset$; $B^{\mathsf{invalid}}_{\mathsf{wcd}} := B_{\mathsf{all}}$; Trial := 0
2:  while Trial $\leq$ MaxTrialNum do
3:    Randomly create a full, encompassing design tree $\Theta$.
4:    NoImproveNum := 0
5:    while NoImproveNum $\leq$ MaxNoImproveNum do
6:      if $\Theta$ is encompassing and WCD($\Theta$) $<$ WCD$_{\mathsf{min}}$
7:        WCD$_{\mathsf{min}}$ := WCD($\Theta$); $\Theta_{\mathsf{min}} := \Theta$
8:        $P_{\mathsf{wcd}}$ := a subset of $P^{leg}$ s.t. $|\mathsf{prefix}(p_1, p_2)| =$
              WCD$_{\mathsf{min}}$ for all $p_1, p_2 \in P_{\mathsf{wcd}}$
9:        $B^{\mathsf{invalid}}_{\mathsf{wcd}} := \bigcup_{p \in P_{\mathsf{wcd}}} B^{\mathsf{invalid}}(p)$
10:       NoImproveNum := 0; Trial := 0
11:     else
12:       NoImproveNum++; $\Theta := \Theta_{\mathsf{min}}$
13:     $q$ := generate a random number in $[0, 1)$
14:     if $q < \epsilon$ then
15:       Randomly select non-terminal block $b_i \in \Theta$
16:     else
17:       Randomly select non-terminal block $b_i \in B^{\mathsf{invalid}}_{\mathsf{wcd}}$
18:     Compute PrunedRegions($b_i$) if it does not exist.
19:     Randomly select a regional vertex $v_j \in V^r$ in $b_i$
          and $v_j \notin$ PrunedRegions($b_i$)
20:     Let $b_{k_1} \in \mathsf{dom}(v_j)$ s.t. $b_{k_1} \in \Theta$.
21:     Randomly select $b_{k_2} \in \mathsf{dom}(v_j)$ s.t. $k_1 \neq k_2$.
22:     $\Theta := (\Theta \setminus \{b_{k_1}\}) \cup \{b_{k_2}\}$
23:     if a design subtree of $b_{k_2}$ was generated previously
24:       Reuse the latest design subtree of $b_{k_2}$ in $\Theta$
25:     else
26:       Randomly generate a design subtree of $b_{k_2}$ in $\Theta$
27:   Trial := Trial + 1
28: return $\Theta_{\mathsf{min}}$
```

WCD. Lines 18 and 19 implement the design subtree pruning rule as described in the paper. The number of random restarts is controlled by MaxTrialNum, which is a given parameter. In each trial, the local search keeps improving Θ_{min} until WCD$_{\mathsf{min}}$ remains unchanged for MaxNoImproveNum iterations, where MaxNoImproveNum is another parameter. The running time of the algorithm depends on how quickly it converges to a local minimum, but typically, the running time is linear to the number of blocks, making it suitable for large-scale GRD problems.

Compact Path Trees

Let $b = \mathsf{parent}[b']$ be the parent of b' and $v_r = v[b']$ be the regional vertex in b that can be substituted by b'. Let $(V, V^r, E, E^r, V^{\mathsf{entry}}, V^{\mathsf{exit}})$ be the specification of b', where $\mathcal{G}' = (V, V^r, E, E^r)$ is the extended search space of b', V^{entry} is a set of entries, and V^{exit} is a set of exits. Let V^{in} and V^{out} be the incoming and outgoing vertices of v_r in b, respectively.

Let $p = \langle v_1, v_2, \ldots, v_m \rangle$ 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 $\mathcal{G}'$ of b' many times or do not go through $\mathcal{G}'$ at all. Suppose p goes through $\mathcal{G}'$ via the regional vertex v_r. We want to identify a subpath p' of p that lies inside $\mathcal{G}'$ entirely.

If there is $(v_i, v_{i+1}) \in \mathsf{edges}(p)$ s.t. $v_i \in V^{\mathsf{in}}$ and $v_{i+1} \notin V$ for some $1 \leq i < m$, v_{i+1} is the first vertex in the subpath p'. If there is another edge $(v_j, v_{j+1}) \in \mathsf{edges}(p)$ s.t. $v_j \notin V$ and $v_{j+1} \in (V \cap V^{\mathsf{out}})$ for some $1 \leq i < j < m$, v_j is the last vertex in p'. Then p' is $\langle v_{i+1}, v_{i+2}, \ldots, v_j \rangle$. After we identify p', we create a *compact* path c by replacing $\langle v_{i+1}, v_{i+2}, \ldots, v_j \rangle$ with the regional vertex v_r in p. That is, $c = \langle v_0, v_1, \ldots, v_i, v_r, v_{j+1}, v_{j+2}, \ldots, v_m \rangle$.

However, suppose (v_i, v_{i+1}) exists where $v_i \in V^{\mathsf{in}}$ and $v_{i+1} \notin V$, but (v_j, v_{j+1}) where $v_j \notin V$ and $v_{j+1} \in V^{\mathsf{out}}$ does not exist. It means that p entered $\mathcal{G}'$ of b' and does not exit $\mathcal{G}'$ through V^{out}. Since the goal vertices are ordinary vertices in the root block, p must have exited $\mathcal{G}'$ via another regional vertex $v_{r_1} \in V^{\mathsf{out}}$, where $v_{r_1} \neq v_r$ and there is an edge from v_r to v_{r_1} in the extended search space of b. Then, we look into the exits of v_{r_1} to see whether p exits from them. If not, we look into the exits of another regional vertex v_{r_2} in the set of outgoing vertices of v_{r_1}, and so on. Eventually, p will exit from one of the regional vertices in b. Let $\langle v'_r, v'_{r_1}, \ldots, v'_{r_l} \rangle$ be the chain of regional vertices p goes through. Although we do not know which section of the subpath p belongs to $\mathcal{G}'$ and which section belongs to the feasible blocks of other regional vertices in the chain, we create a compact path $c = \langle v_0, v_1, \ldots, v_i, v_r, v_{r_1}, \ldots, v_{r_l}, v_{j+1}, v_{j+2}, \ldots, v_m \rangle$.

We need to check whether the remaining of c (i.e., $\langle v_{j+1}, v_{j+2}, \ldots, v_m \rangle$) passes through other regional vertices in b or re-enters some of the regional vertices in $\langle v_r, v_{r_1}, \ldots, v_{r_l} \rangle$. 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 vertice v in c can appear multiple times in c since p can go through v multiple times. To distinguish all instances of a regional vertice 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^{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^{compact} to a terminal vertex of T^{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 v (e.g., C3 in Figure 1), they will share the same trunks in T^{compact} for the vertices in $\mathsf{prefix}(p_1, p_2)$. 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^{compact}.

We construct a compact path tree T^{compact} from a legal path tree T as follows. Initially, T^{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^{compact} will be substituted by v' as well. Moreover, if a subpath p' of p is substituted by a sequence of regional vertices $\langle v'_1, v'_2, \ldots, v'_m \rangle$ in the corresponding compact path, p'

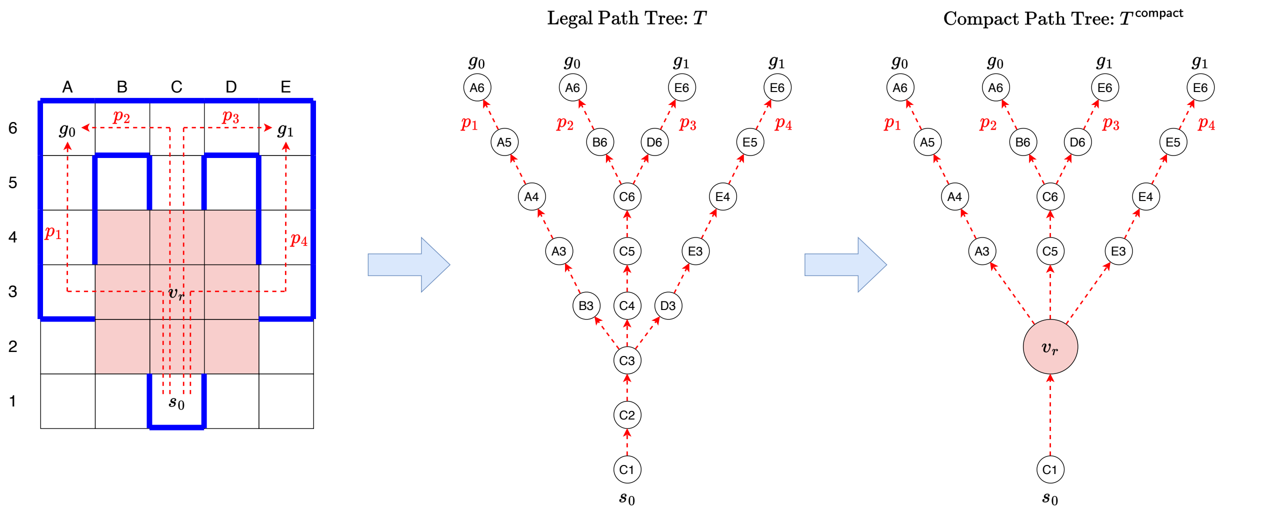
Legal Path Tree: T
Compact Path Tree: T^{compact}
A B C D E
6 5 4 3 2 1
g_0 g_1 p_1 p_2 p_3 p_4 v_r s_0
A6 A5 A4 A3 B3 B6 C6 C5 C4 C3 C2 C1 D6 D3 E6 E5 E4 E3

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^{compact}, which is the same as T except that the set of states inside the regional vertex v_r in T are replaced by v_r.

in T^{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^{compact}. Therefore, we must add a new branch after v' in T^{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^{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_{root} has three regional vertices: v_1, v_2, and v_3. The domains of v_1 and v_3 are $\mathsf{dom}(v_1) = \{b_1, b_2\}$ and $\mathsf{dom}(v_3) = \{b_5, b_6\}$, respectively. These blocks act like on-off switches for the legal paths to g_2 and g_7. The domain of v_2 is $\mathsf{dom}(v_2) = \{b_3, b_4\}$. 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, $\mathsf{dom}(v_2)$ 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 $P = P^{leg}$. 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^{compact} by replacing the subpaths in T that go through the regions with the regional vertices, as described in the previous section. More precisely, $\langle$B9$\rangle$ is replaced by v_1, $\langle$D9$\rangle$ is replaced by v_2, $\langle$E9$\rangle$ is replaced by v_2, and $\langle$G9$\rangle$ is replaced by v_3. The tree at the right side of Figure 2 is T^{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_{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^{min} to compute the lower and upper bounds of the internal vertices in T. T^{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^{min} as shown in the tree in the middle of Figure 3.

Since the remaining four paths in T^{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^{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^{min}. Adding more legal paths to T^{min} could only increase the relative WCDs. Hence, these relative WCDs are the lower bounds of the relative WCDs after the design subtrees are chosen.

For the sake of completeness, for every vertex v in T but not in T^{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

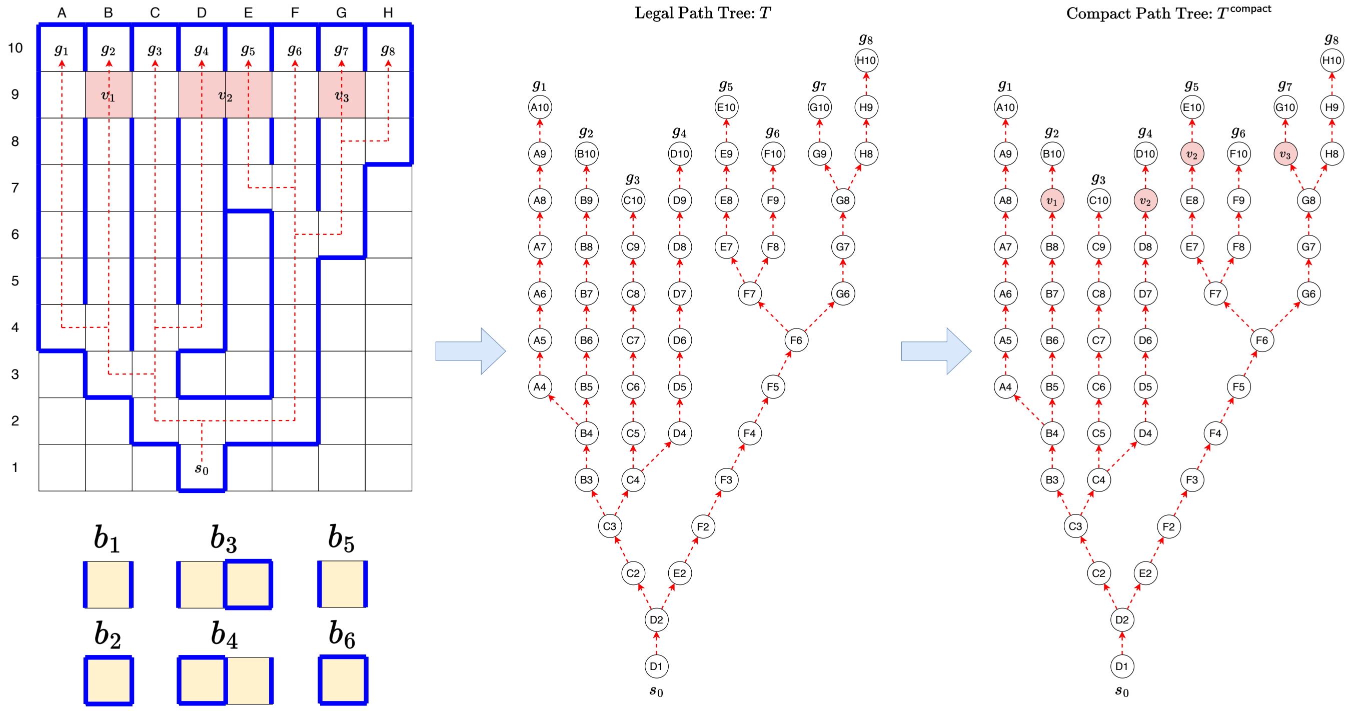

Figure 2: An environment with three regional vertices: v_1, v_2, and v_3, where $\mathsf{dom}(v_1) = \{b_1, b_2\}$, $\mathsf{dom}(v_2) = \{b_3, b_4\}$, and $\mathsf{dom}(v_3) = \{b_5, b_6\}$. There are eight legal paths. The legal path tree T and the compact path tree T^{compact} of the legal paths are shown in the figure.

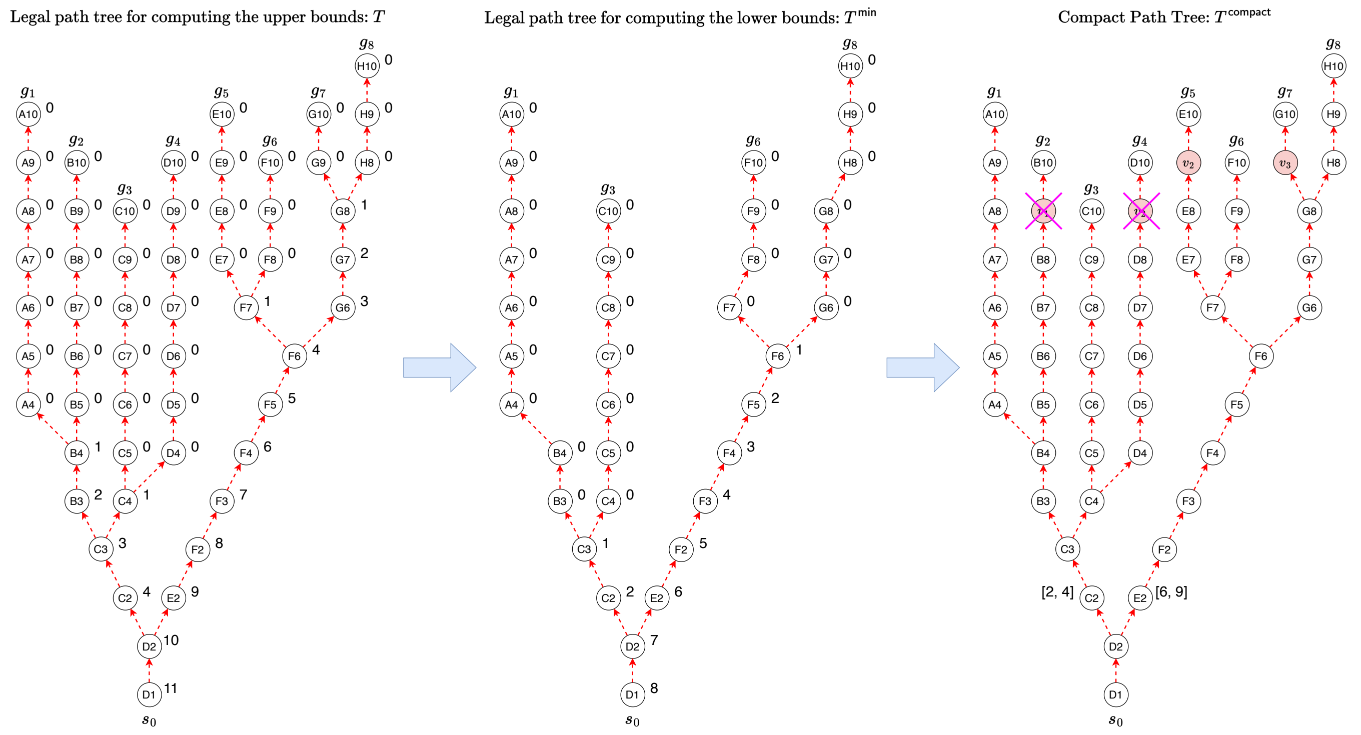

Figure 3: In the legal path tree T, the number adjacent to a vertex is the upper bound of the relative WCD of the state. In the legal path tree T^{min}, the number adjacent to a vertex is the lower bound of the relative WCD of the state. In T^{compact}, the pair of numbers adjacent to C2 and E3 are the lower and upper bounds of the relative WCDs of the state. The two crossed vertices in T^{compact} are marked as "pruned" . However, only v_1 is put in $\mathsf{PrunedRegions}(b_{\mathsf{root}})$, where b_{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 $v_1, v_2 \in \mathsf{children}(v)$ 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 the 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^{compact} as "pruned". The subtree of C2 in T^{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 $\mathsf{PrunedRegions}(b_{\mathsf{root}})$ for the BFS to prune the design subtrees of v_1. We do not put v_2 in $\mathsf{PrunedRegions}(b_{\mathsf{root}})$ because v_2 occurs twice in T^{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 g_5. 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_1 to save time. The pseudocodes of the BFS and the local search algorithm in this document have already utilized $\mathsf{PrunedRegions}(b_{\mathsf{root}})$ to prune v_1.

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 $\mathsf{parent}[b]$ such that s_0 is not an initial vertice 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 $\mathsf{parent}[b]$, we can obtain the lower and upper bounds of the relative WCDs of the child vertices in $\mathsf{parent}[b]$. However, this requires us to compute the lower and upper bounds of the relative WCDs of the vertices in $\mathsf{parent}[b]$ *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, $\mathsf{parent}[b]$ is always evaluated before b. If some of the child vertices are not ordinary vertices in $\mathsf{parent}[b]$, they must be ordinary vertices in some feasible blocks of another regional vertex v' in $\mathsf{parent}[b]$, and v' is adjacent to the regional vertex of b in $\mathsf{parent}[b]$. In this case, we need to evaluate the chain of regional vertices in $\mathsf{parent}[b]$ 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 $\mathsf{dom}(v_1) = \mathsf{dom}(v_2) = \{b_1, b_2, b_3\}$. 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 $\mathsf{dom}(v_1)$ and $\mathsf{dom}(v_2)$ 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 $\mathsf{dom}(v_3) \subseteq \mathsf{dom}(v_1) \times \mathsf{dom}(v_2)$. For example, in Figure 4, we have $\mathsf{dom}(v_3) = \{(b_1, b_2), (b_1, b_3), (b_3, b_2), (b_3, b_3)\}$. Since $|\mathsf{dom}(v_3)|$ is much less than $|\mathsf{dom}(v_1) \times \mathsf{dom}(v_2)|$, the performance speedup is achieved by avoiding the enumeration of all possible combinations of $\mathsf{dom}(v_1)$ and $\mathsf{dom}(v_2)$.

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 $\frac{m^{k(h+1)}-1}{m^k-1}$ 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 $\prod_{v_i \in V^r} \mathsf{dom}(v_i)$ is $\alpha \prod_{v_i \in V^r} |\mathsf{dom}(v_i)|$, where V^r is a group of regional vertices whose child blocks

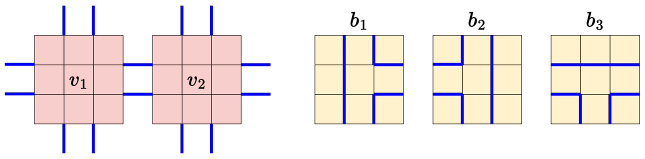

Figure 4: An example of correlated blocks. The regional vertices v_1 and v_2 can be substituted by blocks b_1, b_2, and b_3. If a design constraint is that we must keep the connection between the regions represented by v_1 and v_2 open, some combinations of the substitutions would violate this design constraint.

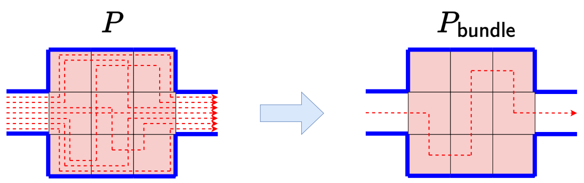

Figure 5: An example of legal path bundles.

are correlated and $0 < \alpha \leq 1$. After combining the correlated blocks, the number of different design trees is roughly $\frac{(\alpha m)^{k(h+1)}-1}{(\alpha m)^k-1}$. If α is 1, the number of design trees remains the same. If α is less than 1, the "branching factor" m^k is multipled by α^k, 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^b is the subpath of p_i that lies inside b entirely, 2) p_i^{prefix} is the prefix of p_i before p_i^b, and 3) p_i^{suffix} is the suffix of p_i after p_i^b. Let $\text{path}[p_i^b] = \text{path}[p_i^{\text{prefix}}] = \text{path}[p_i^{\text{suffix}}] = p_i$. We say p_i is the *complete* path of p_i^b, p_i^{prefix}, and p_i^{suffix}. The collection of these subpaths are stored in $P_b^{leg} = \{p_i^b\}_{p_i \in P^{leg}}$, $P_{\text{prefix}}^{leg} = \{p_i^{\text{prefix}}\}_{p_i \in P^{leg}}$, and $P_{\text{suffix}}^{leg} = \{p_i^{\text{suffix}}\}_{p_i \in P^{leg}}$.

Let $G = \{g_1, g_2, \ldots, g_n\}$ be the set of all goals. We further partition P_b^{leg} into $\mathbb{P}_{\text{goal}}^b = \{P_{g_1}^b, P_{g_2}^b, \ldots, P_{g_n}^b\}$ such that for every $p_i^b \in P_{g_j}^b$, 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_b^{leg} *is* goal-equivalent *if and only if* $P_{g_i}^b = P_{g_j}^b$ *for all* $P_{g_i}^b, P_{g_j}^b \in \mathbb{P}_{\text{goal}}^b$*, where* $1 \leq i < j \leq n$*.*

This definition states that the set of subpaths for every goal in $\mathbb{P}_{\text{goal}}^b$ are the same in a goal-equivalent P_b^{leg}. For any two different goals g_{j_1} and g_{j_2}, if there is a subpath $p_i^b \in P_{g_{j_1}}^b$ s.t. $\text{path}[p_i^b]$ reaches g_{j_1},there is also another subpath $p_k^b \in P_{g_{j_2}}^b$ s.t. $\text{path}[p_k^b]$ reaches g_{j_2} and $p_i^b = p_k^b$.

Likewise, we partition P_{suffix}^{leg} into $\mathbb{P}^{\text{suffix}} = \{P_{p_1^b}^{\text{suffix}}, P_{p_2^b}^{\text{suffix}}, \ldots, P_{p_m^b}^{\text{suffix}}\}$ for all $p_i^b \in P_b^{leg}$, where $P_{p_i^b}^{\text{suffix}}$ is the set of all suffices whose complete paths contain p_i^b. The second assumption for forming legal path bundles is:

Definition 2 P_{suffix}^{leg} *is* suffix-equivalent *if and only if* $P_{p_i^b}^{\text{suffix}} = P_{p_j^b}^{\text{suffix}}$ *for any* $p_i^b, p_j^b \in P_b^{leg}$ *s.t.* $p_i^b = p_j^b$*.*

In other words, if two subpaths in P_b^{leg} are the same, the set of suffices of the corresponding complete paths are also the same.

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

Let $p_{\text{longest}}^b \in P_b^{leg}$ be the *longest* subpath in P_b^{leg}. For each legal path $p_i \in P^{leg}$, 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}^{leg} be the set of legal paths after replacing p_i^b with p_{longest}^b. Obviously, P_{bundle}^{leg} is a subset of P^{leg} since it only includes the legal paths whose subpaths in P_b^{leg} are p_{longest}^b. The following theorem states that if we substitute P_{bundle}^{leg} for P^{leg}, the WCD remains the same.

Theorem 1 *If 1) $p_i^{\mathsf{prefix}} = p_j^{\mathsf{prefix}}$ for all $p_i^{\mathsf{prefix}}, p_j^{\mathsf{prefix}} \in P_{\mathsf{prefix}}^{leg}$, 2) P_b^{leg} is goal-equivalent, and 3) $P_{\mathsf{suffix}}^{leg}$ is suffix-equivalent, the WCD of P^{leg} and the WCD of P_{bundle}^{leg} are the same.*

Proof First, since all subpaths in p_i^{prefix} are the same, the WCD of P^{leg} must be larger than the length of any subpath in p_i^{prefix}. Second, we prove by contradiction that the state that yields the WCD does not exist in the subpaths in P_b^{leg}. Assume the state that yields the WCD is the last state of $\mathsf{prefix}(p_1^b, p_2^b)$, where $p_1^b, p_2^b \in P_b^{leg}$ such that the goals g_1, g_2 of their corresponding complete paths are different, respectively, but $p_1^b \neq p_2^b$. Without loss of generality, let $|p_1^b| \geq |p_2^b|$. We have $|\mathsf{prefix}(p_1^b, p_2^b)| < |p_1^b|$ since p_1^b and p_2^b are different. Since P_b^{leg} is goal-equivalent, there exists a subpath $p_3^b \in P_b^{leg}$ that is the same as p_1^b 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 $\mathsf{prefix}(p_1^b, p_2^b)$ since $|\mathsf{prefix}(p_1^b, p_3^b)| = |p_1^b| > |\mathsf{prefix}(p_1^b, p_2^b)|$. Hence, by contradiction, there are no $p_1^b, p_2^b \in P_b^{leg}$ such that the last state of $\mathsf{prefix}(p_1^b, p_2^b)$ yields the WCD. Third, let s on some paths in $P_{\mathsf{suffix}}^{leg}$ be the state that yields the WCD. Since $P_{\mathsf{suffix}}^{leg}$ is suffix-equivalent, the relative WCD of the first state of any subpath in $P_{\mathsf{suffix}}^{leg}$ are the same. Since all subpaths in p_i^{prefix} have the same length, there exist complete legal paths $p_1, p_2 \in P^{leg}$ where the last state of their common prefix is s such that their subpaths in P_b^{leg} are p_{longest}^b. It turns out both p_1 and p_2 are in P_{bundle}^{leg}, and hence the WCD of P_{bundle}^{leg} are the same as the WCD yields by s in P^{leg}. □

Since the WCD of P_{bundle}^{leg} is equal to the WCD of P^{leg} but P_{bundle}^{leg} is much smaller than P^{leg}, we could substitute P_{bundle}^{leg} for P^{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_b^{leg} so that P^{leg} becomes large, the size of P_{bundle}^{leg} remains the same, and we can use P_{bundle}^{leg} 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 $\mathcal{G} = (V, V^r, E, E^r)$ and 2) a subset $P \subseteq P^{leg}$ 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* $\mathsf{subpath}(p, v_1, v_2)$ *is the subpath of p from v_1 to v_2, inclusively, then $v \in V$ for all $v \in$* $\mathsf{subpath}(p, v_1, v_2)$ *and $(v, v') \in E$ for all edge (v, v') on* $\mathsf{subpath}(p, v_1, v_2)$*) 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 $\{\mathsf{subpath}(p, v_1, v_2)\}_{p \in P}$ can be partitioned into $\mathbb{P} = \{P_{g_i}\}_{g_i \in G'}$ such that $P_{g_{i_1}} = P_{g_{i_1}}$ for any $g_{i_1}, g_{i_2} \in G'$, where $G' = \{g[p]\}_{p \in P}$ 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 $P \subseteq P^{leg}$ of* all *legal paths in P^{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 RWCD(Θ, 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 Θ.*

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 $P \subseteq P^{leg}$ of* all *legal paths in P^{leg} that first go through v_1 and then go through v_2 and has the same prefix before v_1, if P is goal-equivalent and RWCD-equivalent for v_1 and 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_{bundle} for any design tree Θ, where $P_{\mathsf{bundle}} \subseteq P$ is a subset of paths in P whose* $\mathsf{subpath}(p, v_1, v_2) = \mathsf{subpath}(p_{\mathsf{longest}}, v_1, v_2)$*, where* $p_{\mathsf{longest}} = \arg\max_{p \in P} |\mathsf{subpath}(p, v_1, v_2)|$ *is the path with the longest subpath between v_1 and v_2.*

Proof Let us assume RWCD$_1$ at v_1 in the legal path tree formed by P is different from RWCD$_2$ at v_1 in the legal path tree formed by P_{bundle}. It means that there are two legal paths $p_1, p_2 \in P$ that yields RWCD$_1$ at v_1, but $p_1 \neq p_{\mathsf{longest}}$. Due to goal-equivalence, there exists $p_3 \in P$ such that $p_3 = p_{\mathsf{longest}}$ and $g[p_3] = g[p_1]$. Likewise, there exists $p_4 \in P$ such that $p_4 = p_{\mathsf{longest}}$ and $g[p_4] = g[p_2]$. Then RWCD$_3$ at v_1 formed by p_3 and p_4 will be larger than RWCD$_1$, which contradicts the fact that RWCD$_1$ is a relative WCD at v_1. Hence, we have RWCD$_1$ = RWCD$_2$. □

We can apply Theorem 2 to replace a subset of legal paths $P \subseteq P^{leg}$ with P_{bundle} such that $|P_{\mathsf{bundle}}| \leq |P|$ 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

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58(1–3): 161–205.

Sosič, R.; and Gu, J. 1994. Efficient Local Search with Conflict Minimization: A Case Study of the N-Queens Problem. *Knowledge and Data Engineering*, 6(5): 661–668.