House Reconfiguration Problem (HRP)

Kaushik Narayan Ravishankar
Arizona State University
knravish@asu.edu

Problem Statement

The House Reconfiguration Problem (Gerhard et. al. 2011) is an abstract representation of (re)configuration issues encountered in practice. This problem is investigated within the larger framework of reconfiguration challenges in combinatorial optimization and artificial intelligence. The reconfiguration problem finds a path between two specified answer sets that includes each intermediate step as a valid answer set. The goal is to check if one set of answers can be turned into another through a series of continuous modifications. An optimal reconfiguration must adhere to a variety of constraints. If starting from an empty configuration, the reconfiguration problem degenerates into a House Configuration Problem (HCP).

The given version of the problem involves finding an optimal reconfiguration of a house made up of sets of things, cabinets, and rooms owned by individuals. The goal is to find the best placement while adhering to capacity and ownership constraints. The challenge is changing the configuration in a cost-effective manner. Each modification incurs a cost, the total of which must be minimized. To achieve a feasible configuration, the solution must minimize these costs while adhering to all constraints. Answer Set Programming (ASP) enables a programmatic generation of the solution space.

Specifications and Constraints

Specifications

- **Things**: Long things require high cabinets, while small things can fit in either high or small cabinets.
- **Cabinets**: High cabinets occupy two room slots and can hold long or small things. Small cabinets occupy one slot and can hold only small things.
- **Rooms**: Rooms have a maximum of four cabinet slots and are exclusively assigned to one occupant.
- **Occupants**: Each occupant is assigned specific rooms and can only store their things in those rooms.

Constraints

- **Storage**: Each thing must be assigned to one cabinet. Cabinets cannot store things owned by multiple occupants.
- **Capacity**: Cabinets have a maximum capacity (e.g., 5 things per cabinet). Rooms cannot exceed their slot capacity.
- **Reconfiguration**: Transition from a legacy configuration to a new setup, minimizing creation, reuse, movement, and deletion costs.

Project Background

The House Reconfiguration Problem showcases the significance of integrating Answer Set Programming (ASP) with Knowledge Representation and Reasoning (KRR) with the Clingo framework as the underpinning workhorse. ASP's declarative programming is aimed at creating structures for logic problems, by defining the problem domains, weak and hard constraints, as well as assumptions and target metrics. This SAT-guided process not only aims for perfect accuracy, but also the efficiency of problem modelling and solution verification. For this project, the 2019 ASP competition's problem description was first clearly understood and broke down into subparts: the initial HCP and then the full HRP. Going through the CSE 579 course material and the faculty lectures on Canvas helped form a rough approach to the problem. Revisiting the course assignments as well as the Clingo documentation pages was crucial to align the code with ASP best practices. This report highlights key ASP concepts like choice rules, constraints, and optimization, among others, which are made use of in the solution to this problem.

Approach for Solving the Problem

This complex problem requires a methodical approach to it. The approach is structured into three well-defined phases. The initial phase involves getting a detailed understanding of the problem and outlining the specific solution constraints. The second phase involves modeling the HCP special case using Clingo, in which essential predicates are

crafted for the special case. The final phase extends the code for the transformations that are generated for the HRP case and validates against the test cases.

Understanding the Problem

The work began with a thorough examination of the problem statement, focusing on the relationships between things, cabinets, rooms, and occupants. A detailed understanding of the specifications and constraints was developed by identifying atom sets, domains, and mappings that form the core of the problem representation. Constraints such as storage limits, ownership exclusivity, and room capacity were outlined. Additionally, edge cases and provided test cases were analyzed to confirm that no significant scenarios were overlooked.

Modeling the HCP special case

The next step was to model the HCP special case in Clingo. This involved crafting essential predicates to encode the fundamental relationships and constraints, such as assigning things to cabinets, cabinets to rooms, and rooms to occupants. Examples include:

```
personTOthing/2, cabinetTOthing/2,
roomTOcabinet/2, personTOroom/2.
```

Initial code drafts focused on satisfying the constraints while minimizing operations. Example of a room-cabinet limit:

```
:- 5 { roomTOcabinet(R, C) :
cabinetDomain(C) }, room(R).
```

This phase is important to validate the correctness of state, as this serves as the foundation for the generalized HRP problem. Some of the test cases were leveraged to validate the correctness of the implementation, and iterative debugging was conducted to ensure the solution adhered to all specified rules.

Extending from HCP to HRP

With the HCP modeled, albeit not in an entirely robust manner, the code was extended to incorporate transformations required for the HRP case. This mainly included reconfiguration logic to transition from a legacy configuration to a new one while minimizing costs. Predicates were added to model creation, deletion, and reuse operations for cabinets, rooms, and mappings. An example is:

```
1 { reuse(cabinet(C)); remove(cabinet(C)) }
1 :- legacyConfig(cabinet(C)).
```

In this phase, assumptions and implementation details were cross-checked with the sample encoding provided in (Ryabokon, Anna. 2015) to ensure consistency and accuracy. Extensive validation against all the provided test cases followed, confirming that the solution could handle various scenarios while meeting optimization goals.

Implementation Details

The solution is implemented in two primary parts: modeling the HCP and extending it for HRP transformations.

HCP: focuses on solving the HCP as a special case of the HRP, where no legacy configuration is provided. It models the domains, mappings, solution sets, and constraints necessary to determine an optimal configuration.

- ***Domain Sets***: The key domains include `thing`, `cabinet`, `room`, and `person`. Cabinets are associated with thing capacities and space requirements in rooms. Rooms have a limited number of cabinet slots, and each room is exclusively assigned to one person.
- ***Solution Mappings***: The solution requires mapping each thing to a cabinet, each cabinet to a room, and each room to a person. These mappings (`cabinetTOthing`, `roomTOcabinet`, and `personTOroom`) are crafted to satisfy the constraints and ensure consistency across assignments.
- ***Solution Sets and Constraints***: Solution sets define the valid configurations of the problem. Constraints ensure feasibility by enforcing that:
  - Things are assigned to exactly one cabinet.
  - Cabinets adhere to capacity limits, such as a maximum of five things per cabinet.
  - Rooms respect slot limits (`roomSlotUsage`), with a maximum of four cabinet slots.
  - Ownership is exclusive: a cabinet or room cannot contain items belonging to multiple people.
- ***Optimization and Output***: The optimization goal for the HCP is to satisfy constraints while minimizing unnecessary operations or resource usage. The output includes the solution sets and mappings, providing a valid and cost-effective configuration.

HRP: builds on the HCP implementation, introducing transformations and additional constraints to handle reconfiguration from a legacy setup.

- ***Reconfiguration Transformations***: HRP requires transitioning from an existing configuration to a new one while minimizing costs. The solution includes predicates to model operations such as:
  - **Reuse**: Retaining existing elements in the same or resized state (e.g., reusing a small

cabinet as high) with an auxiliary predicate `reuse(X)`.
    - **Remove**: Removing elements no longer required in the new configuration (e.g., unassigning a thing from a cabinet) with an auxiliary predicate `delete(X)`.
    - **Create**: Adding new elements to meet requirements (e.g., creating a new room or cabinet).
- ***Costs for Reconfiguration***: Each transformation is associated with a cost. The updated cost model includes:
    - Creation costs for new cabinets and rooms.
    - Reuse costs, including resizing (e.g., `reuseCabinetAsHighCost`).
    - Deletion costs for removing obsolete elements.
    - Default cost values are applied where explicit input values are not provided.
- ***Optimization and Validation***: The HRP implementation builds on the HCP optimization by minimizing the total reconfiguration cost. The solution is validated against provided test cases to ensure correctness and efficiency in both maintaining the legacy configuration and generating the desired new setup. Outputs include the final mappings, reconfiguration operations, and the minimized total cost.

Main Results and Analysis

The program was thoroughly tested on a series of sample instances of varying complexity to validate the accuracy and efficiency of the proposed solution. The examples in the problem statement were verified for correctness, confirming that the solutions were both correct and optimally efficient. Six distinct test scenarios were executed as provided. The program consistently achieved accurate and optimal solutions. The results are shown below.

clingo hrp_soln.asp hrp_example.asp

Figure 1: output of hrp_example.asp

clingo hrp_soln.asp hcp_example.asp

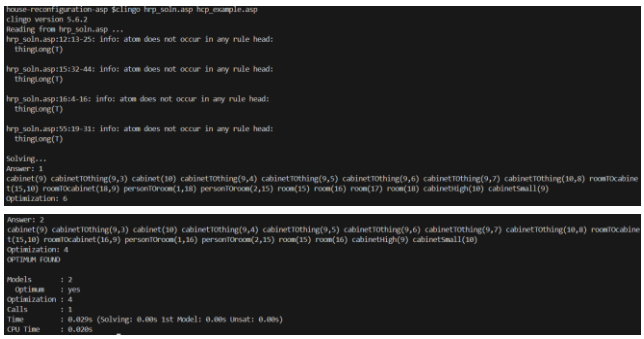

Figure 2: output of hcp_example.asp

clingo hrp_soln.asp emptyconfig_p05t025.asp -t 10

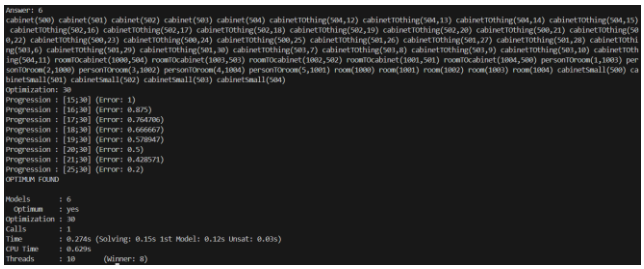

Figure 3: output of emptyconfig_p05t025.asp

clingo hrp_soln.asp longthings_2_p02t030c3.asp -t 10

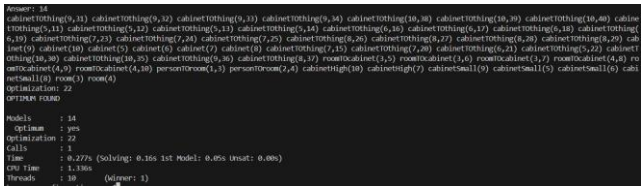

Figure 4: output of longthings_2_p02t030c3.asp

clingo hrp_soln.asp longthings_newroom_p02t024c3.asp -t 10

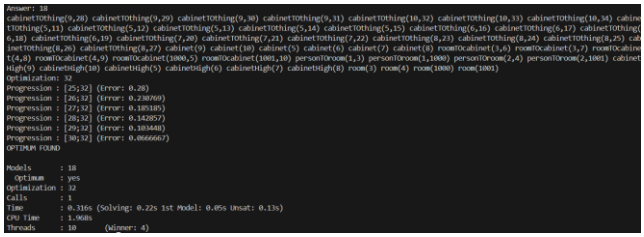

Figure 5: output of longthings_newroom_p02t024c3.asp

clingo hrp_soln.asp switchthingsize_r02t035.asp

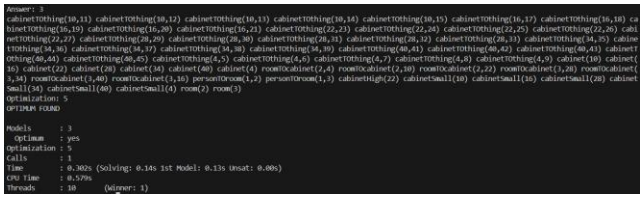

Figure 6: output of switchthingsize_r02t035.asp -t 10

Conclusion

The implementation of the House (Re)Configuration Problem (HCP/HRP) provides a robust framework for generating optimal configurations and performing cost-effective reconfigurations. By leveraging Answer Set Programming (ASP), the solution models the complex relationships between things, cabinets, rooms, and occupants while adhering to strict constraints such as capacity, ownership exclusivity, and storage requirements. The phased approach to solving HCP and extending it for HRP ensures that both foundational configurations and legacy reconfigurations are handled systematically and efficiently.

The results demonstrate the solver's capability to minimize costs through effective use of creation, reuse, and deletion operations, producing solutions that align with provided test cases and edge cases. While the current implementation performs well on small- to medium-sized configurations, future work could explore performance optimization for larger instances, more dynamic cost structures, and additional test scenarios. Overall, the solution successfully meets the objectives of the problem, offering a scalable and adaptable approach to real-world scenarios involving resource optimization and reconfiguration.

Future Work

Future efforts can focus on enhancing the scalability and adaptability of the solution to handle larger and more dynamic configurations. Optimization techniques such as preprocessing to reduce the solution space, heuristic-based prioritization of transformations, or hybrid approaches combining ASP with other optimization methods can improve performance for large-scale problem instances. Additionally, incorporating parallel or distributed computation may significantly reduce solving times for configurations involving numerous things, cabinets, and rooms. Another potential avenue for future work is extending the cost model to accommodate dynamic and context-specific factors, such as variable resource costs or constraints that evolve over time. For example, prioritizing configurations that anticipate future growth or changes in storage requirements could make the solution more practical for real-world scenarios. Finally, expanding the test suite to include more diverse and complex scenarios would ensure robustness and better validate the solution's generalizability to varying problem contexts.b

References

- Friedrich, Gerhard; Ryabokon, Anna; Falkner, Andreas A.; Haselböck, Alois; Schenner, Gottfried; Schreiner, Herwig (2011): (Re)configuration using Answer Set Programming. Proceedings of the IJCAI 2011 Workshop on Configuration.
- Google Sites. 2019. ASP Challenge 2019 Problem Domains. https://sites.google.com/view/aspcomp2019/problem-domains. Accessed: 2024-11-17.
- Ryabokon, Anna (2015): Knowledge-Based (Re)Configuration of Complex Products and Services. Dissertation. Alpen-Adria-Universität, Klagen furt.