USING CASE-BASED PLANNING TO ASSIST LOCAL-SEARCH-BASED PLANNING

HU JUN

NATIONAL UNIVERSITY OF SINGAPORE

2010

USING CASE-BASED PLANNING TO ASSIST LOCAL-SEARCH-BASED PLANNING

HU JUN

(B.Eng, USTC)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF

ENGINEERING

DEPARTMENT OF

ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2010

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Dr. Alexander Nareyek for the continuous support of my M. Eng study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

My sincere thanks also goes to my fellow colleagues in the Interactive & Intelligence Lab and the Games Lab at the National University of Singapore: Amit Kumar, Tian Zhengmiao, Vidal Eric Cesar Jr. Esguerra, Sima Behpour, Huang Manni, for the stimulating discussions, for the hundreds of days working together towards the common objectives, and for all the fun we have had in the last three years. In particular, I am grateful to Amit who offers me great assistance in my research and study. Also I thank my friends who share their lives with me in Singapore: Wang Yang, Li Ti, Zhang Xinquan, Zhu Wanlong, Wang Mengqi, Zhou Yuan, et al.

Last but not the least, I would like to thank my family: my parents Hu Gongmou and Zhang Yuling, for giving birth to me at the first place and supporting me spiritually throughout my life.

Table of Contents

Summary

To pursue a better efficiency of handling complex and dynamic Artificial Intelligence planning problems in real world scenarios, this research is to develop an innovative hybrid planning approach that integrates learning into a general problem solver. There have been case-based planning approaches as well as local-search-based planning approaches in AI planning area in the past. Local-search-based planning is good at handling dynamics in environments by iteratively optimizing the plan quality, while case-based planning plans by learning from previous experiences of problems and situations. A hybrid system that uses local search for the main planning process and applies case-based planning for single improvement iterations seems like a promising idea, because case-based planning is very likely to improve the ordinary planning efficiency and local search can directly serve for the revision phase for the case-based planning step. We review previous approaches for case-based planning, discuss ways to develop the desired case-based planning functionality and its integration into the existing pure local-search-based planning system Crackpot. The initial experimental results indicate that, comparing to the original system Crackpot, the hybrid system confirms its better planning efficiency in both planning speed and planning results. With in-depth analysis, these improvements are very likely resulted from the efficient and effective reuse of the stored knowledge inside the cases. Besides the achievements, several limitations in the current stage are also discovered and some future improvements are planned accordingly.

Keywords: AI planning, learning, hybrid planning approach, case-based planning, local-search-based planning, Crackpot.

1. INTRODUCTION

In this chapter, a general introduction of this research is provided including the motivation, goals and applied methodologies.

Artificial intelligence planning has been an area of research in artificial intelligence for over several decades. It can be thought of searching for a set of actions that can be executed to achieve a goal, e.g., adding the appropriate actions to make a plan that moves a truck to a desired place. The techniques developed in this area have been applied in a variety of tasks including robotics, gaming, process planning, web information gathering, spacecraft mission control, etc.

With years of development, many planning models and planning approaches have been established in the literature for various purposes. However, there are still only limited answers on how to flexibly and efficiently handle planning problems in a dynamic and complex scenario, especially with real-time requirements. Not much research has been done in this area and most of the planning approaches are still working on toy problems.

Our research group, the interactive intelligence lab at the National University of Singapore, is pursuing an innovative, domain-independent planning technology that is capable of handling dynamic and complex planning problems in real world scenarios. We are stepwise improving this planning technology with various features and adapting it to the needs of interactive media. This thesis is going to introduce one such development based on my research work, which is an innovative hybrid planning approach that uses case-based planning (CBP) to assist local-search planning (LSP). This approach is to integrate analogical reasoning as learning into a generative planner to improve its planning efficiency regarding both planning speed and planning results. The initial experimental results support its potential for this purpose.

Local-search planning is an approach that performs planning by iteratively changing plans, e.g., starting with an empty plan and iteratively modifying that plan, by adding, removing or moving actions, etc, to achieve the desired goals. This approach realizes interesting features for planning, such as anytime computation and an uncomplicated handling of an environment's dynamics [26]. Therefore local-search planning is a desirable approach to plan in dynamic and complex environments and it is the base of our planning technology. Now the research group has implemented a prototype of this approach, a local-search planner called Crackpot.

However, pure local-search planning is not sufficient to gain the best performance in handling planning problems in complex and dynamic scenarios. Observations from earlier experiments have revealed several of its deficiencies to be improved. Planning can be thought of a searching through the space of possible sets of resource, e.g., attributes, objects and actions. It is not easy to know which decisions to make to handle the searching effectively if there is not much knowledge available. Planning without such knowledge as guidance is typically computationally expensive, and the time expenses quickly grow when the problem difficulty increases. However, another observation is, the problems encountered in the planning process are not always new ones. When sometimes the planner is asked to solve a problem again, it usually makes one more similar computation and performs no better than it did before. Therefore, to further improve the planning efficiency, using knowledge learning technique to assist the local-search planner seems to be a promising idea.

Case-based planning is a planning approach that has the feature of learning technique. It plans by reusing the remembered experience [13]. It stores the problems experienced and their solutions as cases, and then applies them to solve current problems. The knowledge stored in cases is likely to provide strong guidance in planning, narrowing down the search space by emphasizing some activities and temporarily ignoring the others. Moreover, case-based planning reuses solutions of previous experienced problems as *short-cuts* to construct solutions for future similar problems. This is also helpful to improve the planning efficiency by avoiding the repetition of the same efforts [12, 14, 16, 20 & 35]. Furthermore, case-based planning plans with its past experience simply based on pattern matching technique without complicated analytical process. Therefore, it promises a fast speed in planning. What is more, with high quality cases, CBP has the potential to suggest complex and comprehensive solutions that are expected by target problems, e.g., a desired partial plan with the most wanted Deliever_Package actions to transport a package to its destination. With a careful reuse of them, these case solutions are likely to improve the quality of the overall planning results. Lastly, case-based planning is a relatively independent planning paradigm that can be easily integrated into the existing local-search-based planning system Crackpot.

In conclusion, case-based planning is a learning technique that potentially promises a fast planning speed and high quality planning results. It has the features that just complement our local-search-based planning approach, and it is also applicable to be integrated with the existing local-search planner. Therefore, using case-based planning to assist local-search planning seems an innovative and promising idea to create a hybrid planning approach to pursue a better planning efficiency in complex and dynamic scenarios. This hybridization is a completely new attempt in planning area. And this thesis is going to introduce my research and obtained achievements regarding this topic.

1.1 Goals

The objective of my research is to integrate analogical reasoning as learning into general problem solving to handle planning problems more efficiently. A hybrid AI planning approach is proposed to use case-based planning to assist local-search-based planning. This approach is to be developed and verified by examining its performance in both planning speed and planning results.

As our group has already developed the local-search planning approach and its implementation Crackpot, this research mainly focuses on how to develop the case-based planning approach that best fits the research purposes as well as the existing architecture of Crackpot, and then how to appropriately integrate it into

Crackpot to produce the desired hybrid planning approach. After the establishment of the theoretical models, the designed case-based planner and the hybrid planner are to be implemented and evaluated with practical planning problems.

Moreover, at the first stage, the main goal is to verify this innovative hybrid approach and compare it to the original planner. Therefore, the case-based planner only employs offline learning (use predefined cases) for now and treats online learning (runtime case generation) as another advanced topic. If the hybrid planning approach demonstrates its advantages with initial experimental results, this research will then proceed for further improvements, and the most important objective in the next stage is to employ online learning, which enables the case-based planner to generate and store cases automatically in the runtime. However, the theoretical model of online learning is still built up in this stage, because its design greatly affects the design of the overall CBP process model. A more in-depth research and the implementation of online learning can be expected in the future work.

1.2 Methodology and Structure

This thesis is organized in six chapters according to the research methodology. With literature review, I have studied the related areas such as AI Planning, case-based planning and local-search-based planning. A brief introduction about such background knowledge is provided in the next chapter. By examining the previous approaches and systems in the related areas, I have developed the case-based planning approach that best fits the research purposes, and designed the way to integrate it with the existing local-search planner. The established theoretical model of the case-based planner and the hybrid planner is discussed in chapter 3. After that, the hybrid planner with the case-based planning functionality was implemented for evaluation. The prototype of this hybrid planning system is introduced in chapter 4. This prototype was then tested with practical planning problems and its performance was analyzed by comparing to the original Crackpot system. The experimental results and analysis are presented in chapter 5. Based on the experimental results, some potential improvements are proposed in chapter 6. The overall conclusions and planned future work are discussed in the final chapter.

2. BACKGROUND & ANALYSIS

This research is to develop a hybrid planning approach to pursue a better efficiency in solving AI planning problems. The basic idea is to use case-based planning to assist local-search planning. For a better understanding and explanation, this chapter provides some background knowledge about AI planning, local-search planning and case-based planning. Some analysis is also made along with the literature review.

2.1 AI Planning

Planning is a key area in Artificial Intelligence. In its general form, planning is concerned with the automatic synthesis of action strategies (plans) from a description of actions, sensors, and goals [10]. AI planning techniques have been applied in a variety of tasks including robotics, gaming, process planning, web information gathering, spacecraft mission control, etc.

2.1.1 General Review

AI Planning solves problems in a given domain world, organizing actions to achieve the desired goals, e.g., make a plan that uses a series of actions to deliver a package to a desired place. It involves the representation of actions and world models, reasoning about the effects of actions, and techniques for efficiently searching the space of possible plans.

As Figure 1 illustrates, a typical planning system takes three inputs: a description of the world, a description of the initial state and desired goals, and some possible sensed information about the real world from continuous working sensors. A goal may be a satisfaction, e.g., reach a certain state by a certain time, or an optimization, e.g., minimize the needed time to reach a certain state. All these information are encoded in a formal language, such as STRIPS [9]. The planner produces a plan, an organized collection of actions, to lead from the initial state to a state meeting the goal.

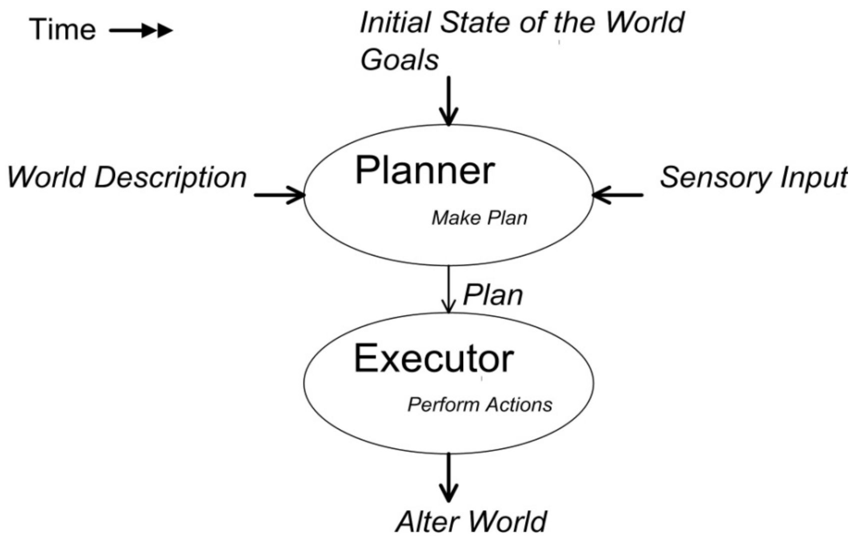


Figure 1: A Typical Planning System.

The world in which planning takes place is often called the *application domain* [15]. The STRIPS terminology is one of the oldest and widely used representations to describe an application domain. A STRIPS representation of the world consists of states, goals, actions, conditions and contributions. States only consist of symbols (no numbers). The state of the world is represented in terms of a set of state variables and their values. A problem is characterized by an initial state and a goal state description. The initial state description tells the planning system the way the world is right now. The goal state description tells the planning system the way the world should be when the plan has been executed. The actions include the components of conditions, what must be established before the action is performed, and contributions, what is established after the action is performed. Example 1.1 shows a STRIPS domain description and a problem instance in Blocks World domain. Logistics domain is another often quoted domain in the literature.

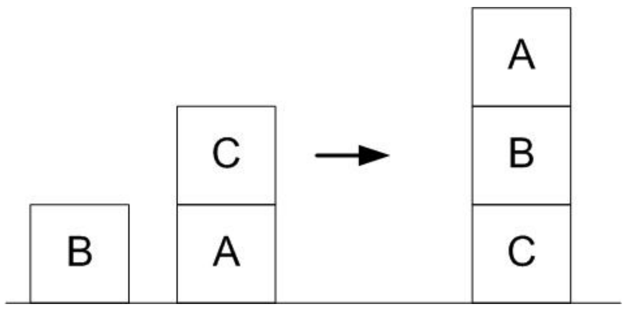
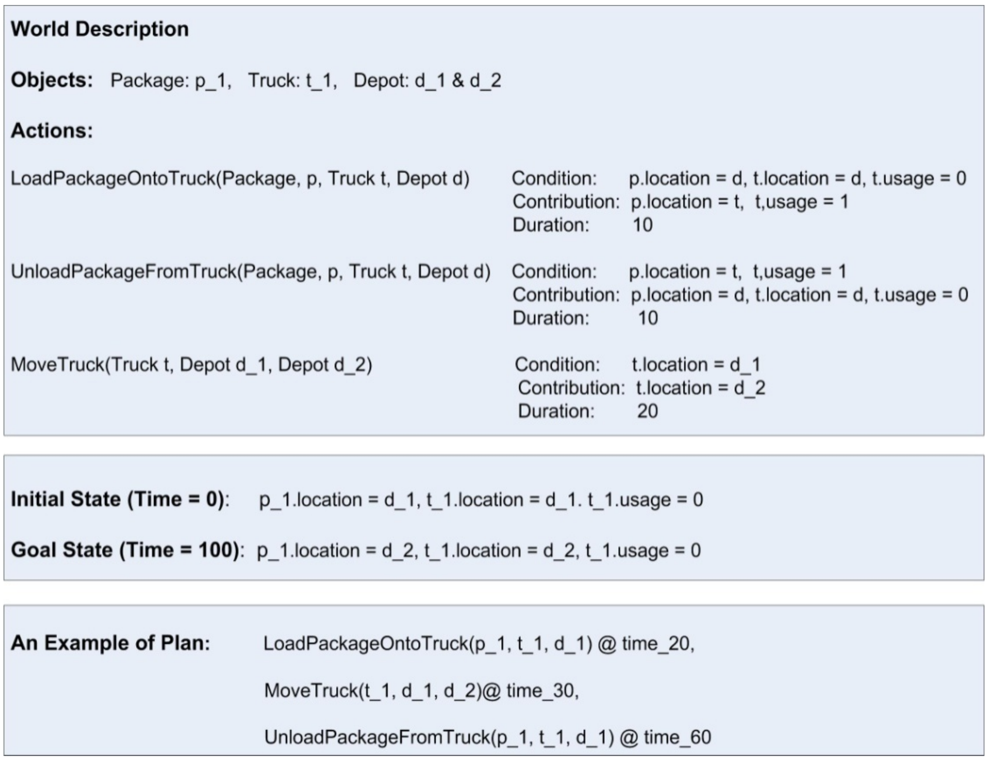


Figure 2: An Example of Blocks World Problem

Example 1.1 See Figure 2. This is a problem in Blocks World domain. The Blocks World domain contains a number of distinct blocks (cubes) of the same shape. In a configuration, a block can be over another block or on the table. In this problem instance, the initial state contains the following set of literals: {Clear(B),Clear(C),On(C,A),On(A,Table),On(B,Table)}. The set represents the conjunction of literals that is all the literals are true. The goal state is {Clear(A),On(A,B),On(B,C),On(C,Table)}. The possible actions are move(block, from, to). The conditions are Clear(block) == true and Clear(to) == true. The contribution has literals ¬On(block, from), On(block, to), clear(from), ¬clear(to) (we assume Clear(Table) is always true).

There are several recent domain representations derives from STRIPS terminology and bring in new features, such as the concept of time, numerical state variables, etc. With the concept of time, goals have been set deadlines to be achieved, actions get durations for execution, and states are allocated on different time points. Figure 3 shows a STRIPS-like domain description for logistics domain as well as a problem and plan instance. It includes time and numerical values.

Planning problems have been handled in two major ways which are, approaches that try to understand and solve the general problem without the use of domain-specific knowledge, and approaches that directly use domain heuristics. In planning, several approaches often use domain-specific heuristics to control the planner's operation. They are called domain-dependent planners, e.g., robot motion planner. Domain-independent planners refer to those which are expected to work for a reasonably large variety of application domains. The goal of our research group is to develop an intelligent, domain-independent planner for real world scenarios.



World Description

Objects: Package: p_1, Truck: t_1, Depot: d_1 & d_2

Actions:

LoadPackageOntoTruck(Package, p, Truck t, Depot d)	Condition:	p.location = d, t.location = d, t.usage = 0
	Contribution:	p.location = t, t,usage = 1
	Duration:	10
UnloadPackageFromTruck(Package, p, Truck t, Depot d)	Condition:	p.location = t, t,usage = 1
	Contribution:	p.location = d, t.location = d, t.usage = 0
	Duration:	10
MoveTruck(Truck t, Depot d_1, Depot d_2)	Condition:	t.location = d_1
	Contribution:	t.location = d_2
	Duration:	20

Initial State (Time = 0): p_1.location = d_1, t_1.location = d_1. t_1.usage = 0

Goal State (Time = 100): p_1.location = d_2, t_1.location = d_2, t_1.usage = 0

An Example of Plan: LoadPackageOntoTruck(p_1, t_1, d_1) @ time_20,

MoveTruck(t_1, d_1, d_2)@ time_30,

UnloadPackageFromTruck(p_1, t_1, d_1) @ time_60

Figure 3: An Example of Logistics Planning Problem with STRIPS-like Representation

Planning is essentially a search problem [15]. The program must traverse a potentially large search space typically about objects, attributes, values and actions, and find a plan that is applicable in the initial state and produces a solution to achieve the goals. This searching can be quite difficult because the search space can contain a large number of elements, complex relations among the elements, and especially many interactions between different states and partial plans. These interactions lead to a surprising amount of complexity; for example, establishing the existence of a precondition in a partially ordered plan can require exponential computation [7], and the problem of finding an optimal plan in even a simple blocks world domain has been shown to be NP-hard [11].

With years of development, there are many planning approaches developed in the literature, and my research mainly focuses on local-search planning and case-based planning.

2.1.2 Local-Search Planning

Local search approach performs a search by iteratively changing an initial assignment of variables. In each of iteration, a *neighborhood* of potential successor states is generated. The quality of the neighborhood states can be computed by a *cost function*. This result is resulted by the *successor choice criterion* to determine the *successor state*. As it is uncertain what kind of change improves a current state, lots of neighbor states are usually analyzed. The *local heuristics*, which encapsulate some domain knowledge, provide some guidance to select the neighbor states as well as the successor. Some of the general work most relevant to local search is on constraint satisfaction, satisfiability testing, and heuristic search [38].

Local-search planning performs planning by iteratively changing fully grounded plans, e.g., starting with an empty plan and continuously improve it by adding, moving or removing actions. In each of iteration, a successor choice criterion determines a plan which will become the new plan. This approach promises interesting features for planning, such as anytime computation and an uncomplicated handling of an environment's dynamics [26].

In contrast to systematic search, which typically searches the whole solution space returning an optimal solution, local search is incomplete (completeness means – if there is a solution, it is definitely found) but more applicable for complex world planning with following reasons.

- The search spaces in planning problems are combinatorial in nature, and therefore are usually huge even for simple-world models. Expressive attributes (like numeric) and complex structures make the search space humungous – the number of actions that can be added into one plan is limited only by the available time and resources; the number of possibilities for an action is bounded only by the multiple of the number of possible objects that take parameterize the action. Therefore, the systematic search approaches that rely on an exhaustive search less likely to success.

- In the real world (for example, in air-traffic control), the utility of the execution of an action, e.g., assigning a base for the plane to land, may be dependent on whether it satisfies a temporal condition specified by an external agent, e.g., a plane in distress must land quickly. Therefore an anytime solution is needed. Our search emphasizes on reaching a reasonable solution as the priority, and only then looking towards optimizing it. This fits well with the local-search philosophy of iteratively improving an early generated solution.
- Systematic searches may not be able to adapt to dynamic changes as they are based on historical information which may become invalid due to the dynamic changes in the world.
- Availability of limited memory may not allow for memory-intensive systematic search approaches.

Generally, for many problems, local search as an algorithm is able to provide approximate solutions that are close (in terms of the objective function) to the global optimum. The average performance of this algorithm in many cases is polynomial with a sufficiently low degree [24]. For example, local search algorithm for the problem of partitioning the vertex set of a graph into two equal parts can have an average-case complexity of $O(n^{2,4})$, where n is the number of vertices.

As a planning approach, the complexity of local-search can be infinite in case of *undecidable* planning problem, which involves parameters with unpredictable bounds [8].

Furthermore, many local-search planning approaches work with fixed maximal plan structures, which contain all possible plan options. Examples are approaches based on Graphplan [6] or SAT [21 & 22]. However, since maximal structures contain all possible options, they can badly scale up in the planning and can hardly be adapted when the environment is dynamic and real-time. One attempt to improve these deficiencies is to use some generative methods, which dynamically change the plan structure during search, e.g., by adding or deleting actions. The example is EXCALIBUR [26].

Our current local-search planner, Crackpot, derives from EXCALIBUR and aims to handle planning problems in a dynamic, complex world with real-time constraints.

2.2 Case-based Reasoning

Case-based planning has grown from a mere application of case-based reasoning (CBR) to a promising approach to solve planning problems [32]. Before talking about case-based planning, there is a brief introduction about case-based reasoning.

Case-based reasoning is a general problem solving paradigm. It uses the specific knowledge of previous experienced concrete problem situation (so called cases) to solve a new problem [1]. It accomplishes this task by finding a similar past case and reusing it in the new problem situation. For example, an auto mechanic who fixes an engine by recalling another car that exhibited similar symptoms is using case-based reasoning. Case-based reasoning is a prominent kind of analogy making.

Case-based reasoning has been widely used for computer reasoning and human behaviour analyzing. There are many CBR systems developed in real life. The prominent examples include SMART, CLAVIER, CoolAir, etc.

2.3 Case-based Planning

Case-based planning is an AI planning approach which inherits the motivations of case-based reasoning. It performs planning as remembering [13]. It solves new planning problems based on the reuse of past experience. This paradigm covers a range of different strategies for organizing and managing past and new problem solving experiences [37].

2.3.1 General Review

A case-based planner can be described as a special application of case-based reasoning in planning area [4]. Its problem areas, working cycle and task-method decomposition are introduced by several authors, e.g., [1 & 18]. The design of a case-based planning typically involves the tasks which can be grouped in the following areas.

- Case Representation. A case in CBP consists of a goal (working situation and target problem) and its solution. Case representation is the task of deciding what to store to form the case and how to organize the memory for effective and efficient case retrieve and case reuse. There are mainly two tasks under this topic, case feature representation and case solution representation. The former one describes the problem and its working situation, which are key factors to identify a case. The latter one stores the solution for that problem based on its working situation.

- Case Retrieve. This task aims at retrieving the case from case memory that best matches the current problem description. In general, it can be decomposed into three subtasks: *feature identification*, *matching*, and *selection* (e.g., ranking) [1].

- Case Reuse. This is the issue of reusing a retrieved case in order to solve the current problem. Its main task is *case adaptation*, which modifies the solution of the retrieved case to better fit the current problem.

- Case Revise. Different from the classical generative planning, case-based planning is able to learn from failures [32]. It includes two subtasks: *evaluation* and *repair*. If failures are detected after applying a case, further repairs will be carried out to validate the solution. And the failure experience are remembered to improve the quality of cases.

- Case Retain. This is the issue of generating new cases for future planning. The new cases are generated either from the ones obtained by revising old cases, or the ones adapted from solutions of new problems. Typically, the retention task involves two subtasks which are *extraction* and *memory arrangement*.

- Case Delete. The CBP system can remove less useful old cases from the case base. It is particularly important to avoid the utility or swamping problem that occurs when case bases grow very large [2]. It helps to make a better use of system memory and maintain a reasonable size of the case base, which is very important for an efficient case retrieve process.

Case representation is the foundation to build up a case-based planning system, and case retrieve, case reuse, case revise, and case retain are the essential tasks to carry out its planning cycle. Their interactions are described by the case-based planning cycle, which is a dynamic model that states the work flow and the evolvement of the system (see Figure 4).

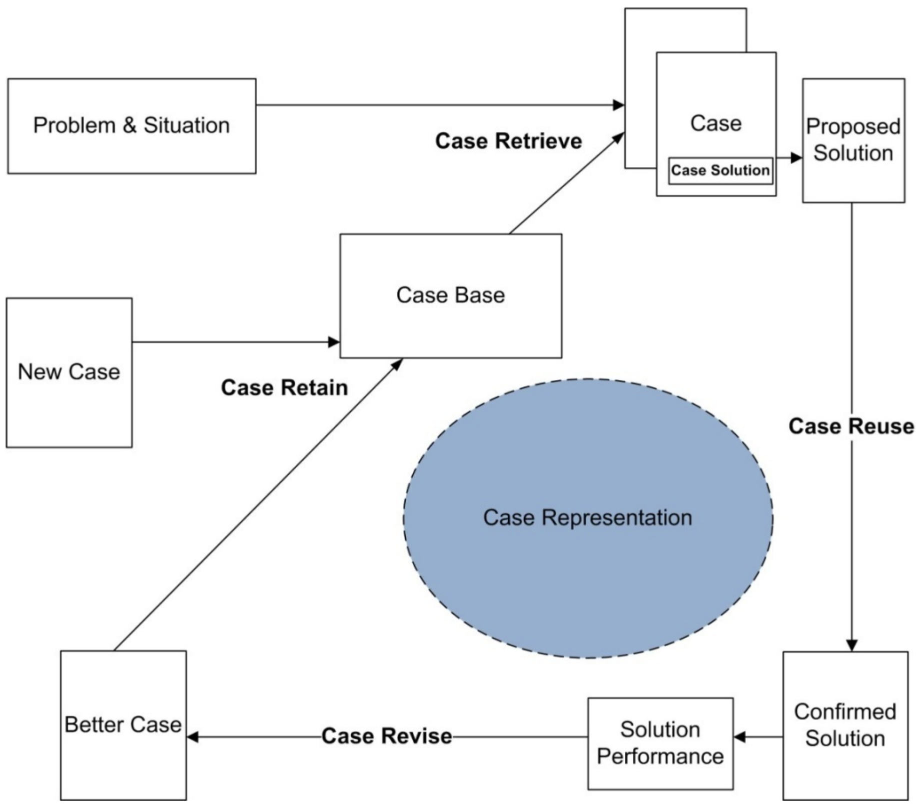


Figure 4: CBP Process Model (adapted from [32])

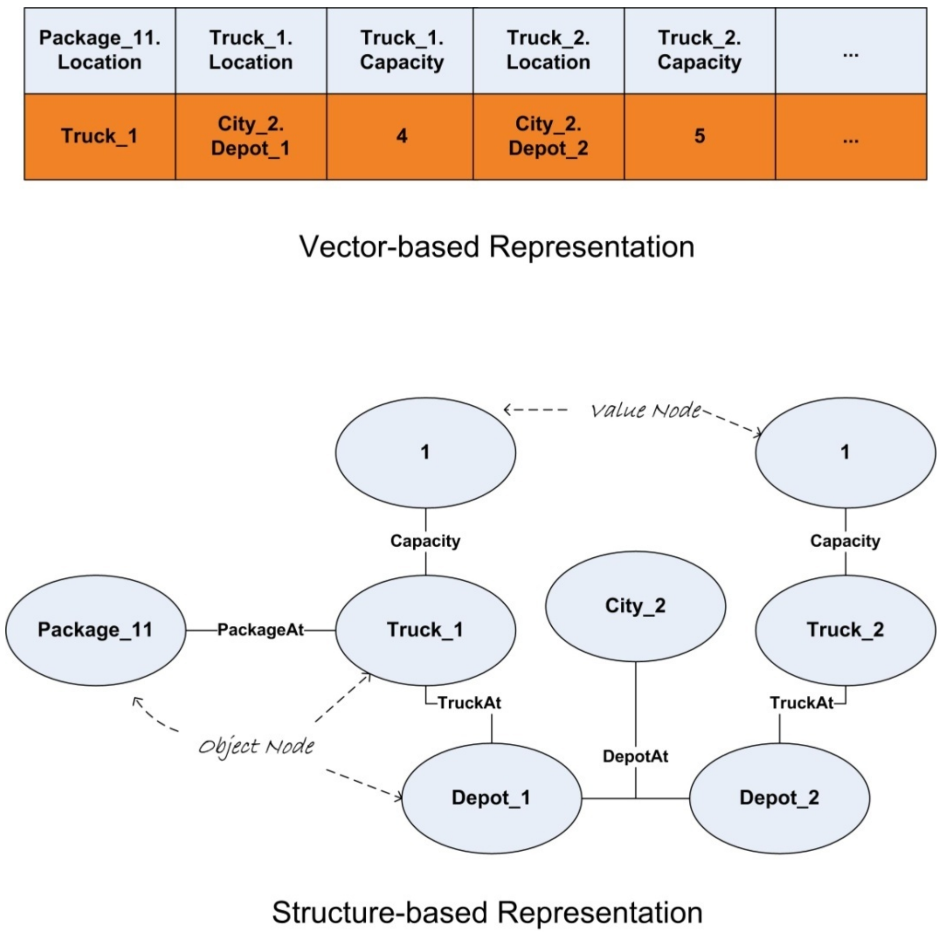
As indicated in [18], the complexity of case-based planning is mainly determined by the planning problem, for example, the number of involved objects, and the case library, for example, the number and quality of cases. Therefore, its complexity can be infinite if the planning problem is undecidable as it involves parameters with unpredictable bounds [8]. Moreover, some empirical results in [4 & 18] indicate, the average performance of a case-based planner also largely depends on how it handles the trade-off of planning efficiency and the size of its case library.

2.3.2 Previous Approaches

In the literature, several case-based planning approaches are developed for various purposes and most of them apply the classic CBP process model in Figure 4. The basic tasks involved in the model are certainly handled in different ways with different context. In this section, we introduce and discuss the major approaches developed and used in the literature for case representation, case reuse, case revise and case retain. They are the essential components to construct a case-based planner.

Approaches for Case Feature Representation

Case feature is part of a case which is used to describe the goal (its target problem and working situation) of that case. How to represent case feature is an important matter which greatly affects the efficiency of case retrieval, and the accuracy of case matching. There are two major types of case feature representations applied in case-based planning: vector-based representation and structure-based representation [5].



Package_11. Location	Truck_1. Location	Truck_1. Capacity	Truck_2. Location	Truck_2. Capacity	...
Truck_1	City_2. Depot_1	4	City_2. Depot_2	5	...

Figure 5: Two Major Types for Case Feature Representation

- Vector-based Representation. One example is the system PROTOS [29]. It expresses facts about individual objects and attributes in a case using independent attribute-value pairs, e.g., the pair (Package_11.Location, Truck_1) in Figure 5. The case features are represented by an arbitrary set of such attribute-value pairs. There are some drawbacks with this representation. Firstly, with a pure enumeration of the attribute-value pairs, it cannot effectively represent the relations between different features, like the relations between different objects. Secondly, it is not efficient and dynamic.

To make up for its insufficient expressivity, this kind of representation usually employs a maximized structure to include all the potential features. The data structure is rigid and also raises a high cost in memory. Therefore, vector-based representation is not applicable to describe cases in a complex domain world, in which there is a large number of elements in the domain and the relations between elements are essential to for the case description. The best advantage of this representation is its linear data structure, which is easy for similarity calculation and case retain.

- Structure-based Representation. One example is the system SCASUEL [25]. It is more expressive by including relations between objects and values. This approach represents features as graphs, in which nodes stand for objects or values, while edges represent the relations between different nodes. The relations can either be the relations between objects, e.g., *is-a*, *has-a* and inheritance, or the relations between objects and values, e.g., a truck and its capacity value. Furthermore, the graph has no fixed size. It can expand out through such relations as bridges to encapsulate new objects and values in the runtime. As a consequence, structure-based representation performs much more expressive and dynamic. However, in the phase of case matching, this complex representation is arbitrarily expensive in computation. According to [31],

a. If case features are represented as unlabelled graphs, matching would be the sub-graph isomorphism problem, NP-Complete.

b. If cases features are represented as labeled graphs, matching cost would be linear in the sum of the number of nodes and number of edges.

c. In practice, the computation complexity lies between the above two extremes.

Approaches for Case Solution Representation

Case solution is another important component of a case. It provides the solution for the target problem which is described by the feature representation. There are majorly two categories of approaches to represent case solutions, *output* and *workflow*. For action planning problems, *output* stands for partial plans, which are segments of the overall plan and comprised of organized actions. Examples are system CHEF [13] and PARIS [2 & 3]. *Workflow* represents the planning decisions made in the planning process, typically comprised of manipulations of actions in action planning, e.g., adding action, removing action, modifying action, etc. Examples are system PORDIGY/ANALOGY [35 & 36] and DERSNLP [16 & 17]. Generally speaking, *workflow* is a more powerful representation than *output*. *Output* can only provide solutions which suggest adding actions into the plan, while *workflow* stands at a higher strategic level and is capable of representing more forms of solutions.

Case representation is the foundation to develop a case-based planning model. Its design greatly affects the thinking of how to construct other parts of the model. For example, case feature representation is closely connected with the phase of case retrieve, while case solution representation is directly connected with the phase of case reuse and case retain.

Approaches for Case Retrieve

Case retrieval is to find the most similar case from the case base using the description of the current problem and working situation. There are mainly three approaches developed for case retrieval. Most CBP systems used to combine different retrieval techniques to exploit their advantages and reduce their limitations.

- Associative Retrieval. It classifies some or all features independently of all the other features [19]. It may be based on a similarity metric, e.g., [23 & 28]. This approach is expensive when the planner must aimlessly compare the given features with each case in the case base which may contain hundreds or thousands of cases.
- Hierarchical Retrieval. It uses features organized into a general-to-specific concept hierarchy, e.g., indexing-based retrieval [19]. The hierarchy can be implemented as discrimination tree or network. It groups the similar cases and thus has the advantage of making easy and fast case retrieval.
- Model-Based Retrieval. It uses a general domain model [1]. This approach model features into an abstract-to-specialized hierarchy. The abstract features

can either be represented by meta-features, which are derived from ground features, or can be represented by introducing some new representing language e.g., system PARIS [2 & 3]. The main problem of this approach is that a considerable amount of general domain knowledge is needed.

Approaches for Case Reuse

Case reuse is to apply the solution suggested by the retrieved case to solve the current problem. The major subtask of case reuse is case adaptation. It is to modify the suggested solution to better fit the target problem. This is one of the most difficult tasks of case-based planning. Consequently this task is performed differently in many CBP systems for different purposes. In general, most of the techniques can be grouped as two categories which are *transformational adaptation* and *derivational adaptation* [32]. They are used to adapt different type of case solutions.

- Transformational Adaptation. It repairs the retrieved case to fix the discrepancies with the current problem. It usually consists of a set of heuristics which directly modify the old solution without changing the case structure. The adaptation operation may be substituting the values of some features, inserting new objects or deleting old ones as well. This method is to adapt the case solutions modeled from *output.*

- Derivational Adaptation. Some recent systems are based on this approach. For this kind of adaptation, the retrieved case is not a concrete plan, but a *guide* that presents how to generate a plan, e.g., system DERSNLP [16 & 17].

It is also called *generative adaptation*. When the case is selected, the system tries to replay it in the new situation to generate a new solution for the target problem. As a consequence, a generative problem solver is necessary for assistance in this approach. This adaptation method is used for case solutions modeled from *workflow*.

Another subtask of case reuse is to apply the confirmed solution, which is adapted from the solution suggested by a case.

Approaches for Case Revise

Case revise is to learn from planning failures and then make further improvements. It contains two major subtasks, *evaluation* and *repair.*

Evaluation evaluates the solution, which is obtained from case-based planning, to identify a failure has occurred or not. There are two kinds of failures: *planning failures* and *execution failures.* Planning failures may arise during case retrieve and reuse when no appropriate solutions are found. Execution failures may occur during solution execution when the solution does not produce the expected result. Case evaluation is usually performed with the two approaches below.

- Evaluation in the real world. The solution is simply executed in the real world and the evaluation carries out along with the execution. However, since it often not acceptable to risk a failure during real execution, most of systems perform evaluation with the second approach.

- Evaluation in the Model. The solution is executed and evaluated in a model-based program, usually a simulation program. How reliable the evaluation is depends on the model adopted.

When a failure is observed, the system may react by looking for a repair. There are two major alternatives for repairs.

- User-repair. The repair is left to the user, e.g., system TOLTEC [33].
- Self-repair. The system tries to repair the failures itself. Some systems exploit the same methods for case adaptation, e.g., system CHEF [13], which applies a set of production rules which can be triggered by failures. Some systems learn from their own past experience. For example, system DERSNLP [16 & 17] retains the repairs with regular cases and replays. The repairs are retrieved using failures and reused with the same case reuse methods.

Approaches for Case Retain

Case retention is to retain the new cases into the case base. The new cases might be generated by revising the existing cases, or adapting new solutions for the new problems. This is the task to improve the knowledge (remembered experience) of the case-based planner. It involves the subtasks of *extraction* and *memory arrangement.*

Extraction is to decide what to use to make a new case and how to use it. It strongly depends on the representation of cases. For *workflow* case solutions, a derivational extraction method is needed, while for *output* case solutions, a transformational extraction is required.

Memory Arrangement is to re-organize the memory of case base while new cases are added in to maintain an efficient case retrieve. There are different strategies for the subtask.

- No arrangement. The case base is left unchanged while the new cases simply added in the memory at the right position. This is simple to accomplish when the case features are sufficiently self-explainable. It is used by most of the case-based planning systems whose case base is already organized in specialization hierarchies, e.g., the cases are classified according to some of its features.
- Arrangement. This is to update and optimize the auxiliary data structures used in the case base, for example, the similarity metric used for measuring case differences e.g., system CHARADE [30], or the external index used for case classification , e.g., system PROTOS [29].

2.3.3 Previous Systems

In this section, four famous and influential case-based planning systems are briefly introduced. They are CHEF [13], PRODIGY/ANALOGY [35 & 36], PARIS [2 & 3] and TOTLEC [33].

CHEF is recognized as the first case-based planning system. It creates new recipes from old ones. CHEF begins planning by finding a recipe that satisfies as many of its active goals as possible. It uses a set of object critics and modification rules to change the old recipe and satisfy the goal of the new one. One of the important aspects of CHEF is explanation of failures through a causal description of why they occurred. CHEF stores new recipes indexed by the goals that they satisfy and the problems that they avoid.

PRODIGY/ANALOGY is a hybrid planning system based on an existing generative planner PRODIGY and it achieves integration of analogical reasoning into general problem solving to solve problems more effectively. Its learning occurs at the strategy level and models *workflow* (planning decision) as case solutions. Consequently, it applies derivational case adaptation approach for case reuse and case retain. Cases are retrieved with a hierarchical retrieve method and case revise is left for the users with a user interface.

PARIS is a domain-independent case-based planning system that introduces abstraction techniques into the case-based planning process. It retrieves, reuses and retains cases at different levels of abstraction. It works as follows. Original cases given at the concrete level are abstracted to several levels of abstraction which leads to a set of abstract cases stored in the case base. Case abstraction is automatically done in the retain phase of CBP cycle. While solving a new problem, an abstract case is retrieved whose abstract problem description matches the current problem at an abstract level. This is the first step. In the following reuse phase, the abstract solution is refined, e.g., the details that are not contained in the abstract case are added to achieve a complete solution of the problem. The refinement is done by a generative planner that performs a forward directed state-space search.

TOTLEC was developed to solve complex manufacturing planning problems such as the detection of errors during the design phase, warning and advising the user about non-manufacturable designs. Its cases are stored in dynamic memory organization. It uses a hierarchical, case-based planning paradigm, a complex indexing of cases, and a three stage incremental retrieval of cases based on the notion of similarity.

3. A HYBRID APPROACH

With background knowledge discussed above, a hybrid planning approach is to be developed to appropriately exploit the advantages of both case-based planning and local-search planning. The local-search planner Crackpot is already implemented and it will be directly utilized. This development is formulated on the design of the CBP approach and the integration of these two approaches. The design is expected to best meet the research objectives and fit the existing system architecture of Crackpot.

3.1 General Approach

Figure 6 provides a global view of the hybrid planner, including the design of the case-based planner and its integration into the local-search planner.

The local-search planner is the main planning process in the work. It owns all the planning resource, e.g., the domain world knowledge and the plan etc., and manipulates them to generate plans to solve the indicated problems.

The design of the case-based planner follows the classic CBP process model (see Figure 4) which proves the most applicable and is widely inherited by many CBP systems. Therefore, our case-based planner also constructs its workflow with the tasks of case retrieve, case reuse, case revise and case retain.

For the integration, the case-based planner is designed to be a plug attached to the main system. It is relatively independent and it can be asked for services when the main system decides to trigger it.

When the case-based planner is working, the main system is responsible for providing the required information, e.g., the information about the current problem, or the available objects in the domain. With such information, the

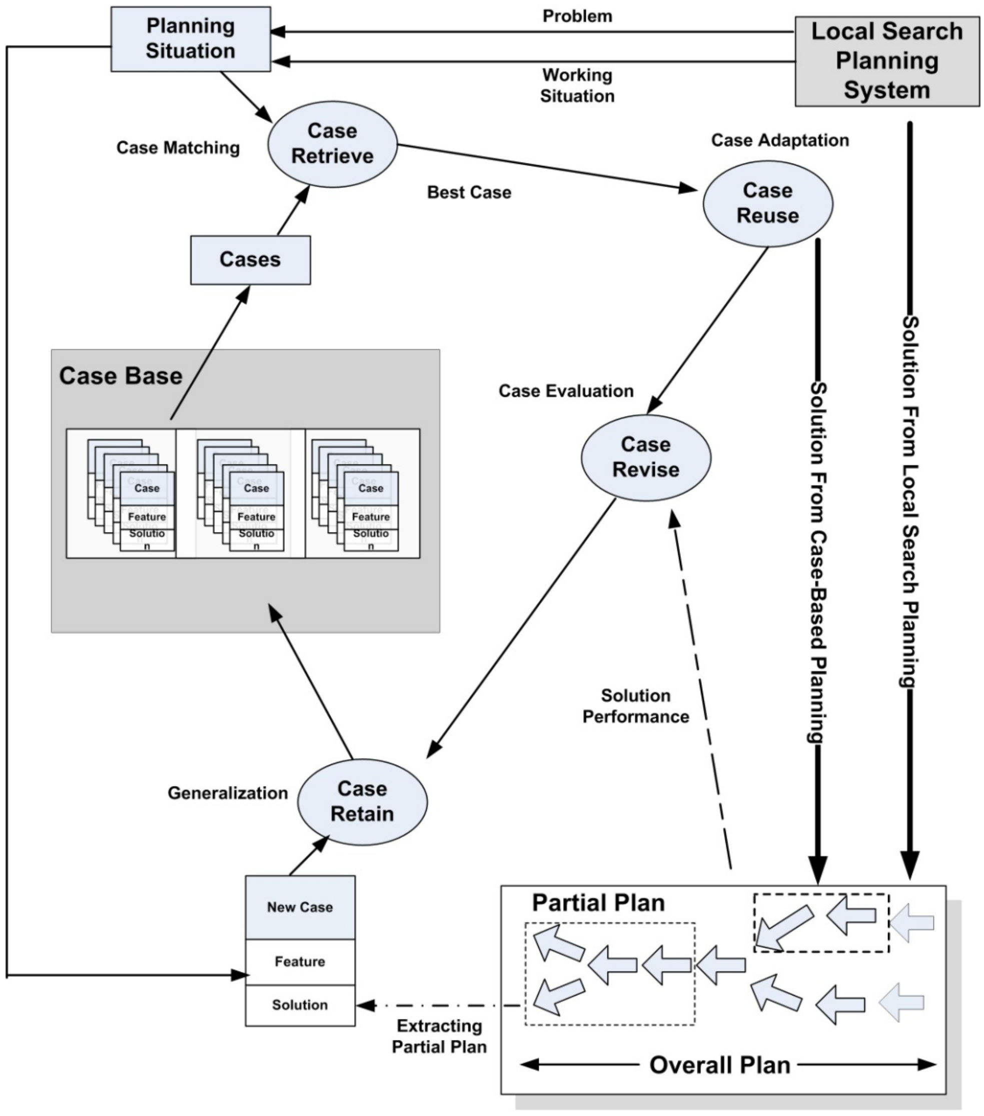


Figure 6: The Global Design of the Hybrid System

case-based planner first finds out the best matching case for the target problem and working situation in the phase of case retrieve, and prepares its solution based on the retrieved case in the phase of *case reuse*. If that solution is estimated as good enough, the case-based planner will submit it to the main system, which is responsible for making the changes suggested by the solution, e.g., adding an action into the plan. After that, the main system will evaluate the applied solution based on the effects it resulted in. The outcome of the evaluation will be used for *case revise,* which is to adjust the *case preferences*. Case preference indicates the usefulness of a case. If the solution provided by a case turns out to be beneficial for the planning, the preference value of that case goes up, otherwise the preference goes down. The cases with higher preference will be more likely to be maintained in the memory, while the ones which are least preferred will be deleted to save space for newly incoming cases, which are continually generated in the phase of *case retain*. For generating and storing new cases in the runtime, the case-based planner also requires assistance from the main system.

3.2 Sub Tasks to Be Accomplished

In general, the case-based planner is to be constructed with the tasks of case representation, case retrieve, case reuse and case retain. Each of these sub tasks requires the most appropriate approach to be accomplished, in order to best meet the research objectives and also be compatible with the existing system. Some of the approaches might be adapted from existing ones studied from literature review, while some of them can only be created to satisfy our unique requirements.

3.2.1 Case Representation

A case in the case-based planner (see Figure 7) is composed of a feature part, which identifies the usage (target problem and working situation) of a case, and a solution part, which solves the target problem described in the feature part. Naturally, there are two sub tasks in case representation: case feature representation and case solution representation.

Case Feature Representation

As studied before, structure-based representation (graphs) is more expressive and flexible but expensive in similarity computation. In contrast, vector-based representation can be easily compared but it has a rigid structure and is short in representing the concept of relations in the domain world, e.g., package_1 is on truck_2 which is at the same city with truck_1. Planning in complex and dynamic scenarios, the relations between objects as well as the relations between objects and values, are very important to accurately describe problems and situations. For example, in logistics domain, truck can only be moved between depots which are in the same city where the truck is. If only given depots, cities and trucks, without specifying how they are connected to each other, the planner cannot tell whether a move between two depots is really applicable or not.

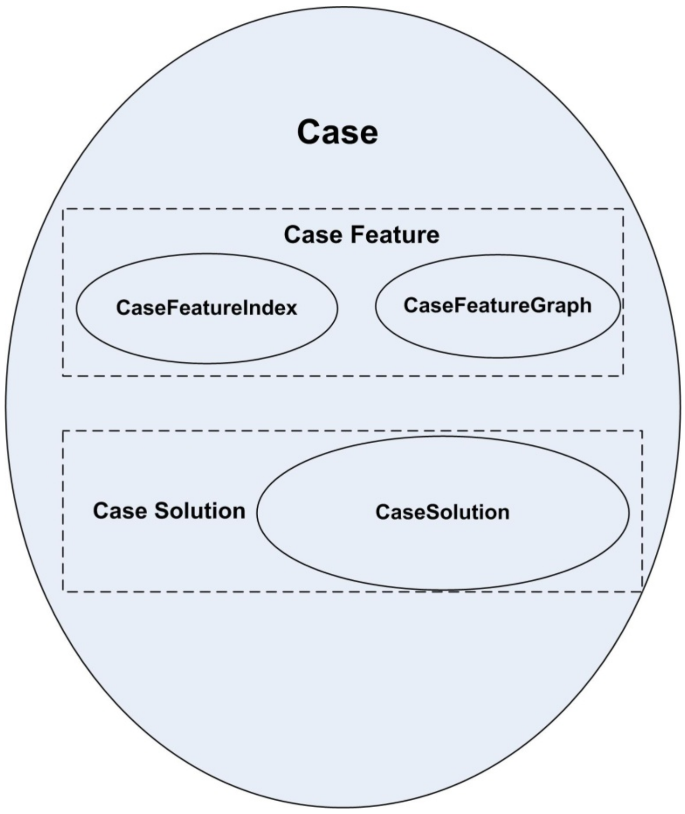


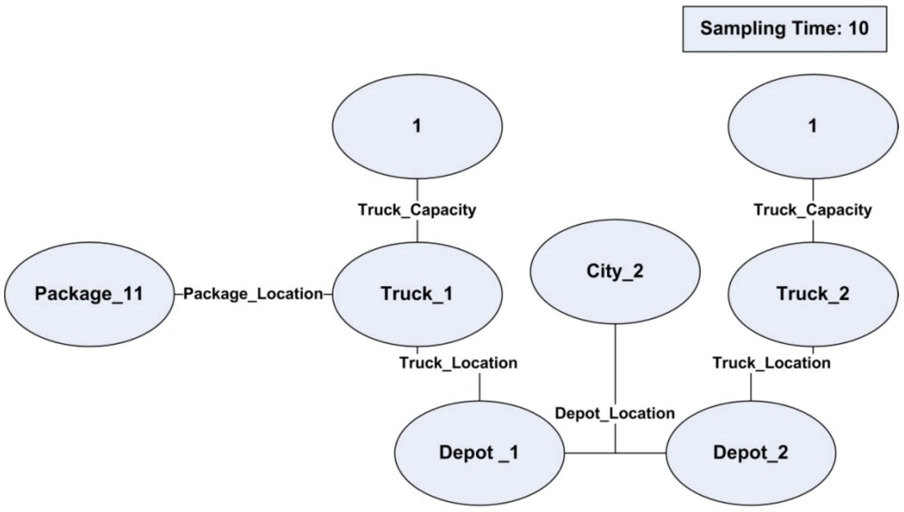
Figure 7: A Case in the Case-based Planner

To express such relations in the domain world, structure-based representation is more desirable. Furthermore, the rigid and maximized structure of the vector-based representation is not applicable to handle dynamic environments where elements, e.g., objects and values, are allowed to be deleted or inserted. Therefore, the structure-based representation is preferred in our case-based planner to describe case features.

Meanwhile, to reduce the large efforts spent on graph matching in case retrieve, the case-based planner utilizes vector-based representation as indices to guide the process of case retrieve.

Problem Type	Graph Information	Length of Solution
Package_Location	Map(Level, ObjectNumber)	100

CaseFeatureIndex



CaseFeatureGraph

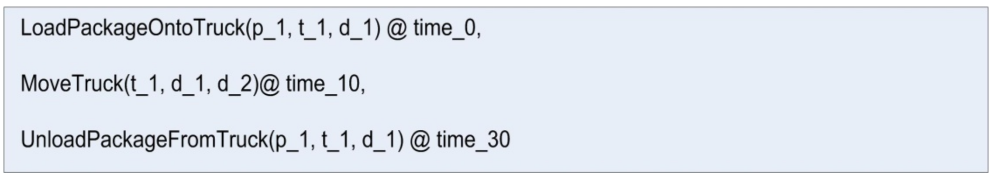
Figure 8: Two Feature Representations in the Case-based Planner

As shown in Figure 8, the feature part of every case contains two elements, a structure-based *CaseFeatureGraph* and a vector-based *CaseFeatureIndex.* The *CaseFeatureGraph* represents the features which describe the case's target problem and working situation. There are two types of nodes inside *CaseFeatureGraph*: object nodes and value nodes. They are connected to each other by object-object edges or object-value edges. Besides this graph, there is another vector-based structure called *CaseFeatureIndex*, which contains very limited but crucial information about the case, e.g., the type of its target problem, the duration of its solution, and some abstract information about its *CaseFeatureGraph*. *CaseFeatureIndex* is the index of a case and its content is simple and always available. Therefore, it is suitable to be represented with vector-based representation for fast matching. While retrieving a case, a simple comparison on the *CaseFeatureIndex* is able to filter out most of the cases that are not similar enough. Only the cases that passed the index matching are eligible to take an in-depth matching on *CaseFeatureGraph*. By further comparing the abundant information inside *CaseFeatureGraph*, it is easy for the planner to identify the most similar case from the candidates.

In conclusion, this two-level case feature representation exploits the advantages of the two representation approaches, and it is expected to make an efficient and effective case retrieve.

Case Solution Representation

Theoretically speaking, *workflow* and *output* are both applicable for making *case solutions* within the existing architecture of Crackpot. The case-based planner can either store the planning decisions, which manipulate actions, or the partial plans, which is comprised of actions obtained by such manipulations. In the current stage, we choose *output* (see Figure 9) to model *case solutions* because of the following reasons.



```
LoadPackageOntoTruck(p_1, t_1, d_1) @ time_0,

MoveTruck(t_1, d_1, d_2)@ time_10,

UnloadPackageFromTruck(p_1, t_1, d_1) @ time_30
```

Figure 9: An Example of Case Solution Modelled with *Output*

1. Limitation of the existing system. As studied before, *workflow* is more expressive and flexible than *output*. A solution of *workflow* can suggest multiple forms of solutions, e.g., adding actions, removing actions, modifying actions, moving actions, etc. In contrast, a solution of *output* can only suggest adding actions, which is only applicable to solve a smaller range of problems. *Workflow* is generally a better option than *output.* However, our case-based planner is built up based on the existing local-search planning system Crackpot. With the current system architecture, some *workflow*s, e.g., removing action, moving action and modifying action, are hard to make be reused in a valid way. For example, when the case-based planner suggests removing one action to solve a problem, this decision is made purely based on the working situation and the target problem. The selected case does not know and also cannot check whether the action to be removed is really in the plan or not. If there is no such action, this invalid removal cannot be performed and the suggested solution becomes useless. Even though occasionally some similar actions exist in the plan, it is also difficult to identify which action is the one that the solution really wants. An invalid or wrong removal operation is not helpful and can be even harmful to the overall planning process. Similarly, moving action and modifying action also have this kind of problem. In contrast, adding action into the plan is always applicable and has much lower chance

being an invalid operations (sometimes happens because of invalid parameter binding). As a consequence, except for adding new actions, other types of *workflow* are difficult to be effectively reused in the current system. Therefore, using *workflow* as case solution actually can do nothing more than using *output*.

2. The consideration of online learning. Online learning means automatic runtime case generation. This is another important feature of the case-based planner. As case solution generation is part of case generation, a careful study is taken to examine which case solution representation is more applicable to be generated automatically.

   In Crackpot, *workflows* are planning decisions, which arrange and manipulate actions, made in iterative repairs. In local-search planning, a problem can be decomposed to many sub-problems and the original problem could eventually be solved by solving all of these sub-problems one after another. An example is given in Example 3.1.

   The decomposition typically carries out through action-condition-action chains and it can go further until no new sub-problems generated. Each of the produced sub-problems will be solved through different *workflows* in different planning iterations because one iteration makes only one piece of *workflow*. In current Crackpot, when one problem is decomposed in this way, the produced sub-problems are not explicitly linked to each other. The system treats them all as individual problems to be solved. As a result, the case-based planner has no idea which sub-problems have the same origin and thus it doesn't know

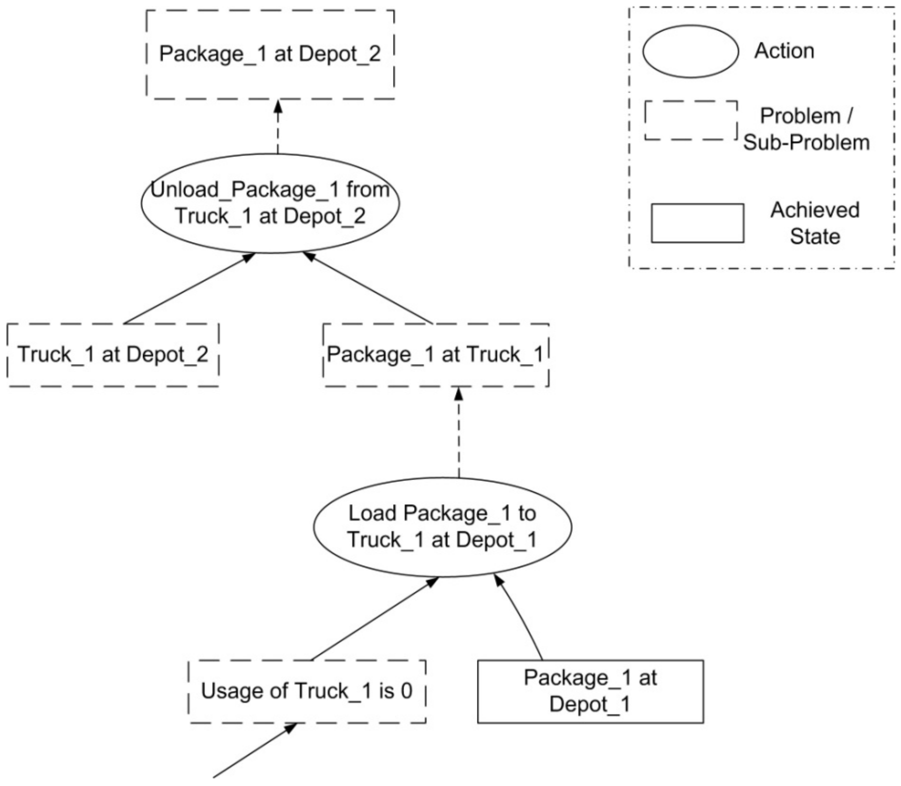


Figure 10: An Example of Problem Decomposition in Crackpot

Example 3.1 As shown in Figure 10, the problem, which delivers package_1 from depot_1 to depot_2, can be decomposed into sub-problems including, make truck_1 empty, make truck_1 at depot_2, make package_1 at truck_1, etc.

which *workflows* need to be collected together to form a complete solution. Moreover, in current Crackpot, the problem to be solved in the next iteration is determined purely according to its severity that how bad this problem is. This is a process without in-depth analyzing on the relevance between problems, e.g., Figures 11 gives a possible sub-problem solving procedure in Crackpot.

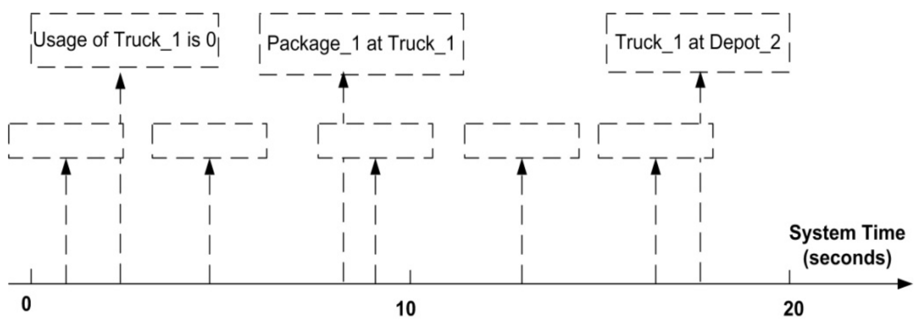


Figure 11: A Possible Sub-problem Solving Procedure in Crackpot for Example 3.1.

Consequently the *workflows* which work on the common objective are not necessarily neighboring to each other, but more likely to scatter in the time span of the whole planning process. To make a single case, the case-based planner must watch the planning process all the time because it has no idea when the next piece of useful *workflow* will come out. This process is computationally expensive. Therefore, even if the current system is improved to identify the relevance between sub-problems, it is still hard for the case-based planner to collect the related *workflows* or even part of them. In contrast, a case solution modeled with *output* can be easily obtained as it is just part of the overall plan.

Based on above analysis, *output* is more desirable to represent case solutions in this case-based planner.

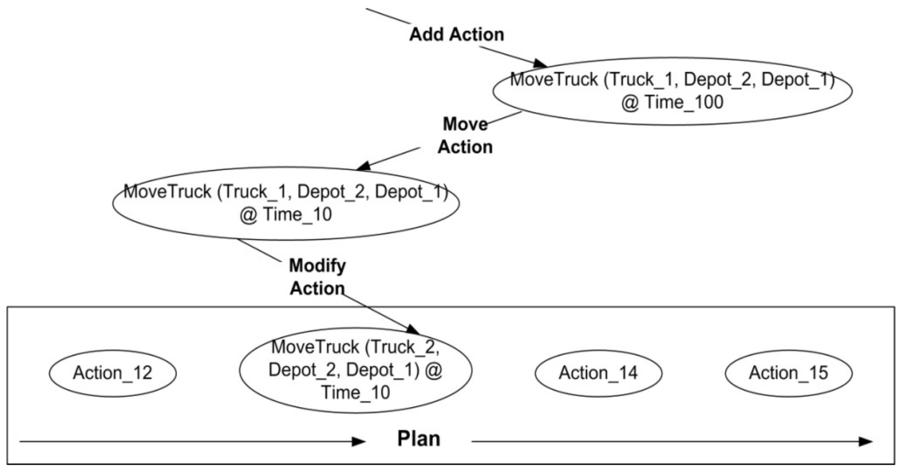

Add Action
MoveTruck (Truck_1, Depot_2, Depot_1) @ Time_100
Move Action
MoveTruck (Truck_1, Depot_2, Depot_1) @ Time_10
Modify Action
Action_12
MoveTruck (Truck_2, Depot_2, Depot_1) @ Time_10
Action_14
Action_15
Plan

Figure 12: One Action in the Plan Obtained from Multiple Workflows

Moreover, sometimes one action in the final plan might be resulted from several steps of *workflows*. For example, in Figure 12, the action, Move_Truck, is eventually finalized in the plan by using three *workflows*. A case solution modeled with *workflow* will store all of these steps, while a case solution of *output* only stores the finally result. Therefore, the form of *output* is more effective and compact.

3.2.2 Case Retrieve

Case Retrieve is first to identify the current problem as well as the working situation, and then find out the most matching case from the case base. As illustrated before, by using vector-based *CaseFeatureIndex* along with structure-based *CaseFeatureGraph*, a hierarchical case retrieve method is applied. The task of case retrieve has two major tasks which are information extraction and case matching.

Information Extraction

Information extraction is the first step of case retrieve. It is to generate a new case (with an empty case solution) to paint an image of the current target problem and working situation. This special case (named as *PlanningSituation*) contains one *CaseFeatureIndex* and one *CaseFeatureGraph*, which need to be filled with concrete planning information.

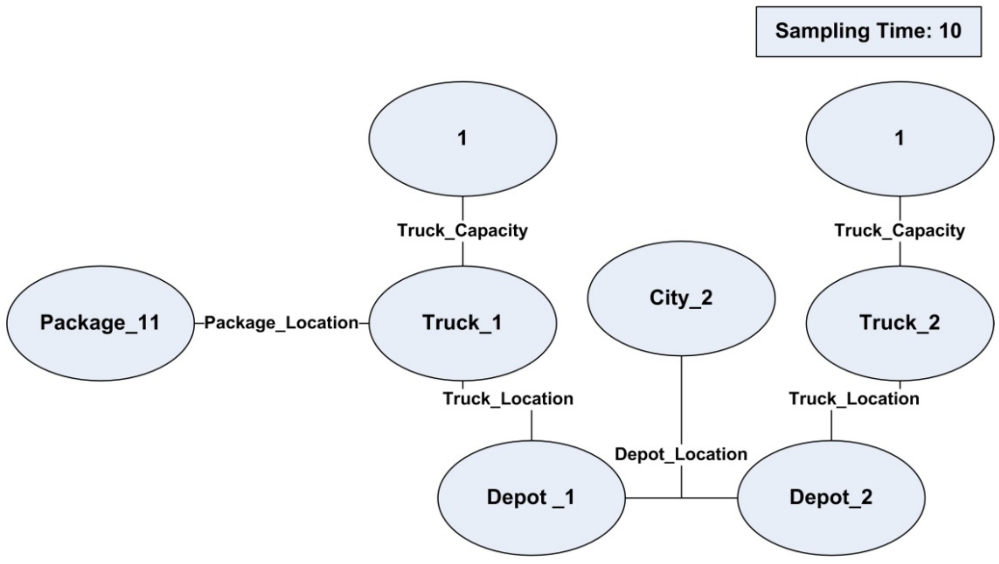


Figure 13: An Example of *CaseFeatureGraph*

In the design, the information extraction starts from one node (one object) and is stepwise expanding through object-attribute-value relation chains. The detailed process is explained with the *CaseFeatureGraph* in Figure 13 as example.

When the main system triggers the case-based planner, it always tells the case-based planner what the current target problem is. In the example, the target problem is about the location of package_11. So the graph adds in package_11 as the first node and starts expanding from this node. But before that, there is another issue to be solved: the sampling time. Crackpot is a planning system with real-time constraints. The problems and situations are all located on the time line, e.g., truck_2 is at depot_1 at time_10 and wants to depot_3 at time_50. Therefore, the case-based planner has to determine a time point based on which to depict the working situation, e.g., to collect the information that where the truck_2 is at time_10 or time_40. In the current design, random selection is applied. This is not the most intelligent way but it proves a workable idea for the initial stage. The only rule for selecting a sampling time is quite general. The time value cannot be negative and it has to be prior to the deadline by which the problem must be solved, because the solution executes from that time and its duration is certainly more than zero (see Figure 14). This general rule tries to make sure a space, which is from the sampling time to the problem solving deadline, to place the solution.

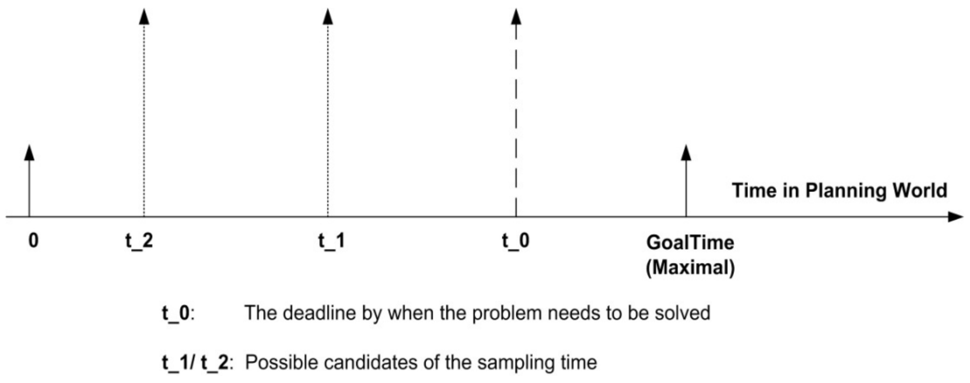

Time in Planning World
0
t_2
t_1
t_0
GoalTime (Maximal)
t_0: The deadline by when the problem needs to be solved
t_1/ t_2: Possible candidates of the sampling time

Figure 14: An Example of Sampling Time

Once the sampling time is determined, the graph expansion starts. The CBP system keeps querying information from the main system through object-attribute-value chains. In the main system, given any two elements of the tuple (object, attribute, value), the third one can be answered. In the *CaseFeatureGraph* in Figure 13, with package_11, its attribute package_location and time_10 (the sampling time in the example), truck_1 can be found because the package_11 is just placed on that truck at time_10. Therefore, the graph adds in an object-object edge, connecting package_11 and truck_1. In *CaseFeatureGraph*, every object node is marked with a level number, which is determined by the object's occurring order in the graph. In the example, package_11 is marked as level_1 because it is the first object in the graph. Truck_1 is marked as level_2 because it is observed through package_11, which is a level_1 object node. Similarly, with truck_1, its attribute truck_location and time_10, the graph adds in depot_1, which is marked as level_3. Simultaneously, with truck_1's another attribute truck_capacity and time_10, the *CaseFeatureGraph* obtains one value node representing truck_1's capacity, and also brings in one object-value edge connecting truck_1 to its capacity value. With similar steps, this graph gradually expands to encapsulate any reachable information as its nodes and edges. This expansion will stop when there is no new node observed or the indicated maximal object node level is reached. To describe a case, the more the data is, the better the precision is. But for data processing, the smaller the graph size is, the faster the speed is. General observation reveals that, the most useful information is what is most closely related to the core of the problem. Therefore, it is reasonable to stop the expansion at a particular expansion level, to make a balance between description accuracy and processing efficiency. The maximal expansion level number is domain-dependent and it can be indicated as planning context input.

In addition, to make cases more general, the stored cases are represented using object parameters and value parameters. Before matching takes place, the real instances of objects and values that are used in the generated *CaseFeatureGraph* will later be translated into object parameters and value parameters that are used by the stored cases. During this translation, the bidirectional mapping between the parameters and real instances is built up.

Case Matching

Case matching is to pick up the most similar case from the case base by comparing them to the special case (called *PlanningSituation*) generated in the step of information extraction. As illustrated before, the feature part of a case is composed by one *CaseFeatureIndex* and one *CaseFeatureGraph*. With a hierarchical case retrieve design, the task of case matching is decomposed into two major levels which are, index matching and graph matching. The detailed workflow of the task is given in Figure 15.

The generated *PlanningSituation* is compared to the cases in the case base. Only the cases which have succeeded in the phase of index matching are eligible to take graph matching. In index matching, there are three steps, problem type matching, solution length matching and graph information matching. In the first two steps, the requirements are very strict. If the case's target problem is different from current problem, or the duration of the case's solution is longer than available length of time,

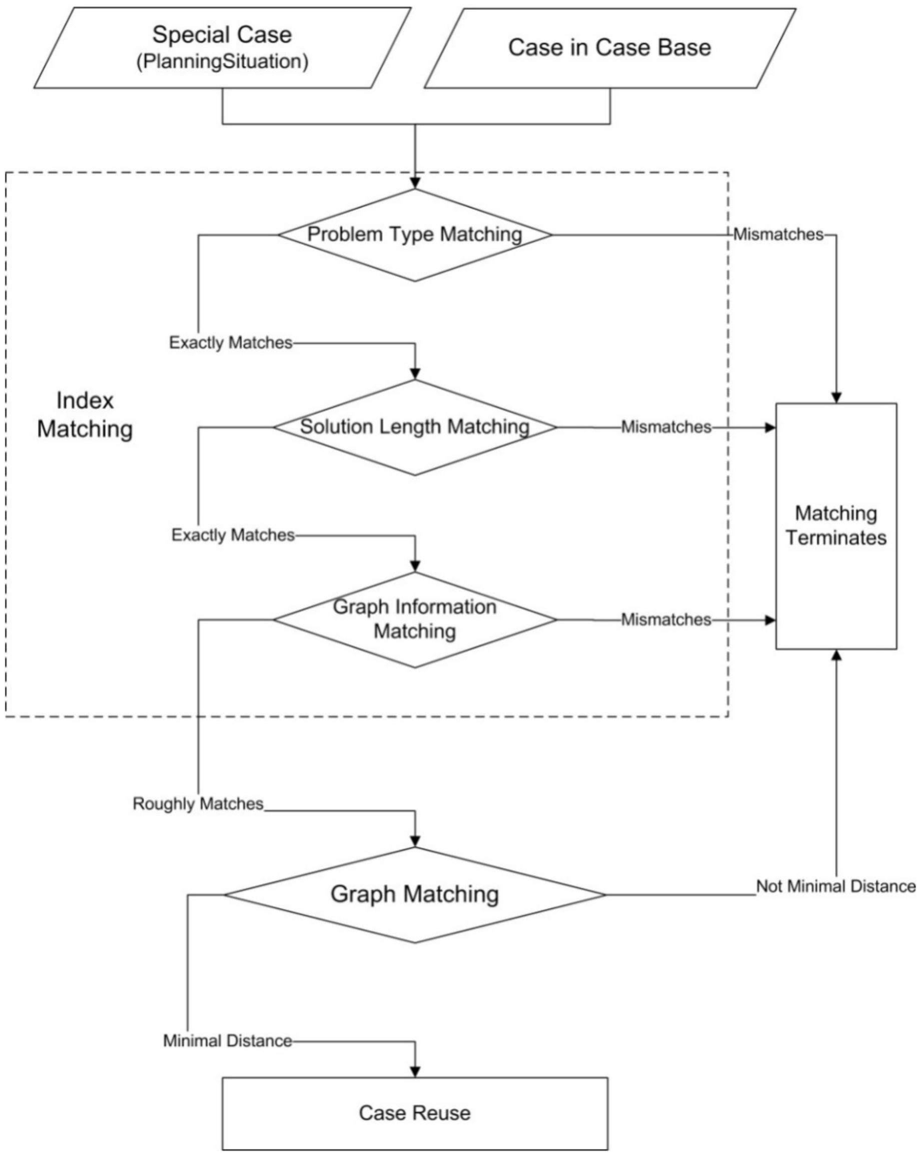


Figure 15: Flow Chart for Case Matching

that case will be given up immediately. In the step of graph information matching, the requirement becomes looser and a distance value is calculated. The graph information is a mapping from object node level to the number of objects which are at that level in the *CaseFeatureGraph*. With the information extraction technique, this simple mapping effectively reflects the overall structure of the graph. The distance of two mappings is defined with equation (1).

$$Index_{Difference(m1,m2)} = \sum_{1}^{m2.MaxLevel} |m2[i] - m1[i]| * LevelWeight(i) \qquad (1)$$

In equation (1), *m2* represents the mapping in the case and *m1* represents the mapping in the *PlanningSituation*. The comparison always starts from the case side because cases always contain the minimal amount of essential information while *PlanningSituation* is designed to take slightly over-sufficient amount of information. The level weight is descending from level one to the maximal level, e.g., a weight of 9 for level 1 and a weight of 3 for level 7. The thinking behind is that, the objects that are close to the problem object are likely to be more important to describe the problem. The CBP system calculates the difference between two such mappings with equation (1). If the result is too large to be acceptable, that case will be given up, otherwise the case becomes a candidate for graph matching.

In the phase of graph matching, *CaseFeatureGraphs* will be compared. The distance between such graphs is defined based on object-attribute-value tuple. The calculation is done by equation (2).

$$Graph\_Difference(g1, g2) =$$

$$\sum_{Object\ i\ in\ g2} \sum_{attribute\ j\ for\ i} Value\_Distance(g2\_value(i,j), g1\_value(i,j)) \quad (2)$$

In equation (2), *g1* represents the *CaseFeatureGraph* in the *PlanningSituation* and *g2* represents the *CaseFeatureGraph* in the case. Similarly, the comparison starts from case side as cases have less but more compact information. The value distance computation is based on the existing attribute value distance computation in Crackpot. (The distance between two values is based on the minimal number of transitions required to transit one to another. The details can be found at [26].) If the object *i* is not found in *g1*, the value distance function will return a large distance value as penalty.

As a result, the case difference is obtained based the index difference and graph difference with equation (3). The *GraphWeight* is larger than the *IndexWeight*.

$$Case\_Difference(p, c) = Index\_Difference * IndexWeight + Graph\_Difference * GraphWeight \qquad (3)$$

Finally, the case with the minimal total difference value will be selected for case reuse.

3.2.3 Case Reuse

Once the most similar case is retrieved, it will be reused to solve the current problem. The first step is case adaptation. Since we have chosen *output* as case solutions, it is natural to apply *transformational adaptation* approach to perform this task. In this case-based planner, the main task for case adaptation is case solution instantiation. For a general usage, the stored cases are all represented with object and value parameters. Therefore, before the retrieved case is applied, its solution must be instantiated with real instances of objects and values in the domain world. As introduced before, we have already built up a bidirectional mapping between real instances and parameters when translating the generated *PlanningSituation*. Now it is time to use this mapping to replace the parameters with real instances.

However, for many reasons, it happens that some parameter is not included in the mapping. In this case, the system has two options. First, it can abandon that part of solution, e.g., one or more actions which use that parameter. Second, it can search for an appropriate instance from the domain world to continue the instantiation, e.g., using any available truck_11 to replace the parameter truck_a. In the current stage, some observations indicate that if a parameter is not used in *PlanningSituation*, the part of case solution which uses that parameter is usually not needed for the current problem, e.g., if an airplane parameter is missing in *PlanningSituation*, it probably means that the current problem asks for delivering a package within a city and an airplane is not necessary.   Therefore the first option is the default method for now. The second method is also used to handle some special cases.

In most of the time, even the most matching case can have slight difference from the current problem and working situation. Therefore, the solution suggested by a case is typically not optimal: some required part might be missing or some existing part might be redundant. The missing part can be repaired in the phase of case revise, but the redundant part is better to be pruned way. In the current design, the case-based planner only identifies the redundant part of solutions by analyzing the unmapped parameters. Some improvements can be expected in the future work.

After the case solution is instantiated, it will be passed to the main planning system to be applied.

3.2.4 Case Revise

Case revise is learning from planning failures. The main task of case revise in our system is to evaluate the usefulness of cases and improve their quality. After a case solution is applied, the system will check the outcome and adjust the preference of the case accordingly. Each case has been associated with one preference value which indicates its usefulness. If the solution suggested by the retrieved case fails to improve the planning process, the preference of that will be decreased. The failures caused by the case solution will be validated by future repairs assisted by the local-search planner. If the case solution turns out to be beneficial, the case's preference will be increased or stay the same (if it already reaches maximal). In the phase of case base management, the cases with the lowest preference values will be eliminated, while the cases with higher preference are more likely to be maintained. This is to produce an efficient case base with high quality cases, and minimize the extra memory consumption caused by the use of case-based planner.

In one case-based planning iteration, which include case retrieve, case reuse and case revise, the complexity of this CBP approach is O(NML), where N represents the number of objects in the domain, M represents the number of object relations in the domain and L is the number of cases in the case base. Therefore the computational complexity very much depends on the complexity of the planning domain as well as the size of the case base. A larger collection of cases might be helpful to provide a better solution but might reduce the planning efficiency.

3.2.5 Case Retain

An adaptive case-based planning system usually features the capability to generate new cases in runtime. This is also our main objective in the next stage of research. There are two major questions to be answered to accomplish this task: when to do case retain and how to generate useful cases in by the system itself.

As Crackpot is also utilizing parallel processing technology, case retain is designed to go parallel with the planning process. While the planning keeps running in the front stage, the runtime case generation carries out in the back stage. Case generation involves case feature generation and case solution generation. Case feature generation is already done for making *PlanningSituations*, while case solution generation is a challenging task to be implemented in the next stage. The basic thinking is illustrated with the example in Figure 16.

To generate a case solution, the case-based planner first selects a problem which might be most frequently encountered or it is relatively completely solved. In the example, a problem about the package’s location is selected. To identify which actions in the overall plan are working on this problem, the case-based planner will backtrack through contribution-action-condition chains as each action makes its contribution to fulfil goals or other actions’ conditions. These links enable the system to reach any actions connected in this way. In the beginning of case retain, the system determines a sampling time according to some policies and then generate the new

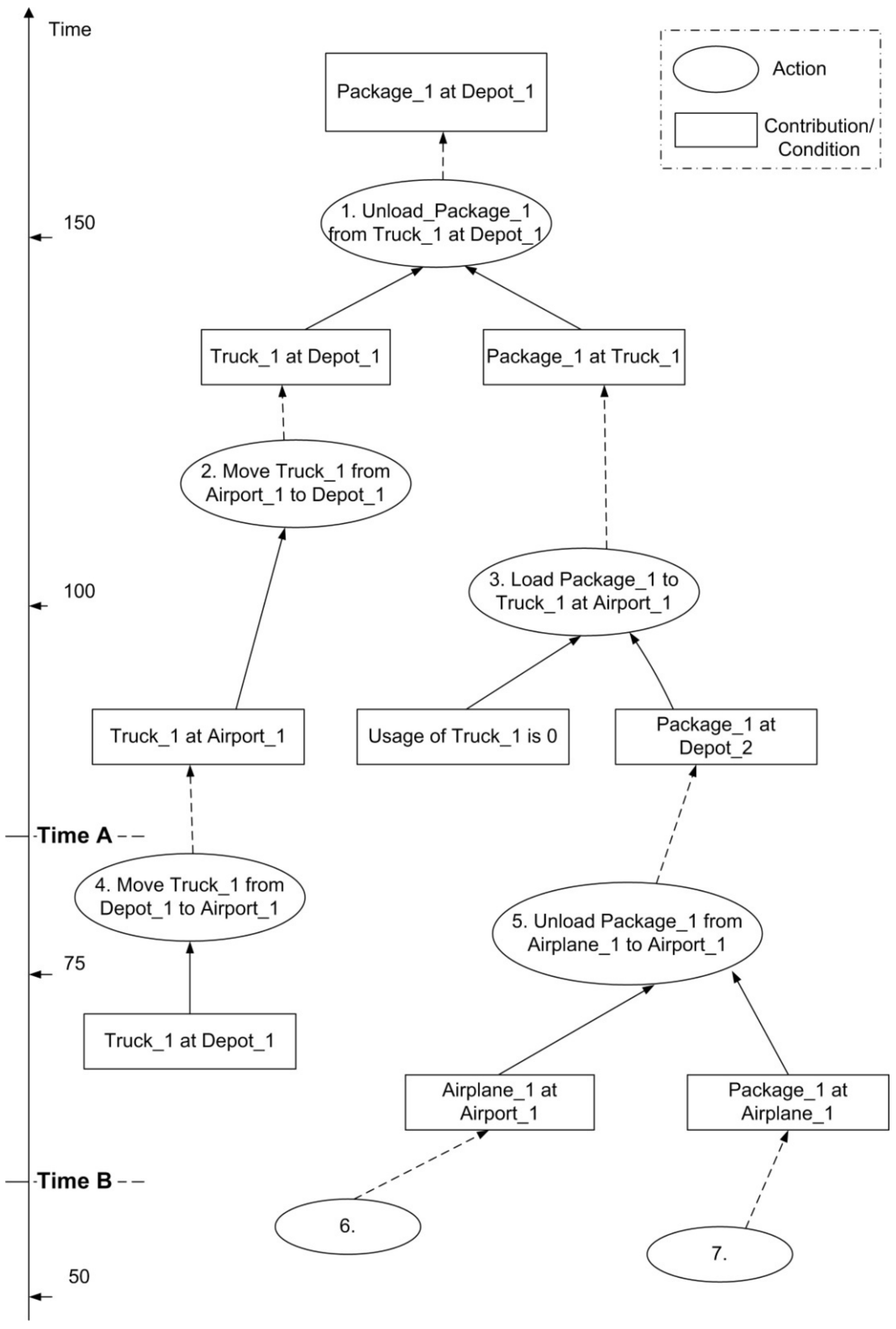


Figure 16: A Piece of Overall Plan for Case Retain

case based on the sampling time. In the example, if time A is selected, action 1, 2, 3 will be included to form the case solution, and the new case will be about how to deliver a package from one depot to another. If time B is selected, action 1 - 5 will be used to form the case solution and the new case will be about how to deliver a package from an airplane to a depot. Depending on the context, either of these cases can be useful.

However, in real planning, even the overall plan itself is incomplete and inconsistent most of the time. The case solutions derived from a part of the overall plan might also be incomplete and inconsistent. Consequently the quality of newly generated cases varies much from one to another. This is one of the major reasons that why we introduce the tasks of case revise and case base management. The low quality cases, which are useless or even harmful, need to be eliminated from the case base to forbid the dangerous reuse of them.

Newly generated cases will be retained into the case base followed by a sequence of memory arrangements. In the end of planning, all the cases remained in the case base will be written into files as the result of learning.

3.2.6 Case Base Management

Any computer system tries to optimize its memory use. With runtime case generation, new cases are continuously added into the case base. The case-based planner should develop policies to manage the case base to make a best of memory.

Moreover, it is also important to keep a relatively small case base for an efficient case retrieving process.

The basic objective of case base management is to make a compact case base, filled with the most useful cases. The quality of the cases also reflects the learning capability of the case-based planner. As illustrated before, every case is now associated with a preference value which is determined by the performance of the case. The cases which are least preferred are likely to be removed. Some improvements to better manage the case base can be expected in the future work along with other new functionalities of the case-based planner.

3.3 Integrating CBP into LSP

The objective of this research is to use case-based planning to assist local-search planning. In this prototype, local-search planning is the main process, while the case-based planning is plugged to the main process through a local-search heuristic called case-based planning heuristic (CBP heuristic). The local-search planner triggers the case-based planner by calling the CBP heuristic to take planning tasks. Similar to the other local-search heuristics, the CBP heuristic is assigned with one preference value. The higher its preference value is, the more likely it will be chosen to take a task. The preference value will be updated according to the heuristic's performance in the planning and a high preference value indicates a better reliability of that heuristic.

Furthermore, this case-based planner models *output* as case solutions and it directly adds in new actions (usually multiple actions) in one iteration to improve the overall plan. There is already another similar local-search heuristic employed by Crackpot, which adds in single action in one iteration. CBP heuristic is just like an updated version of the existing one. Therefore, model case-based planner as a local-search heuristic is the most straightforward and applicable way to integrate it into the local-search planner.

3.4 The Innovation of Our CBP System

Comparing to the traditional case-based planning systems, such as CHEF [13], PRODIGY/ANALOGY [35 & 36], PARIS [2 & 3] and TOTLEC [33], the main differences of our CBP system are, the structure of case feature representation, and the supports from a local-search based planning system.

Firstly, in our CBP system, case features are represented by graphs, which possess not only the individual objects (as vertices), but also their relations between (as edges). For example, when depot_1's is at city_1, city_1 is not only a value in the representation but another object which is connected to depot_1 via the edge of depot_location. In the meanwhile, city_1 as an individual object can also be connected to other objects through different object relations. By including object relations, this structure allows our case-based planner dynamically grasp the most useful information to make features in the runtime, while some traditional systems always have to keep a fixed maximal structure to contain all information.

Secondly, the LSP system is helpful to revise the old solutions to fit the current problems. In case adaptation, traditional CBP systems can only make small changes based on previous solutions, while our system can have more constructive changes because of the LSP system. Some desired adaptations might be difficult to be made by the CBP system, for example, adding new airplanes to transport crowded packages, but they are relatively easy for the LSP system because it is a generative planner. These unsatisfications will raise costs in the LSP system and are probably to be handled by the LSP system eventually.

Thirdly, traditional CBP systems can only learn to solve the problems which are similar to previous problems. The newly remembered cases are always adapted from the old ones with small differences. If there is a new problem that has no reference in existing cases, a traditional CBP system normally cannot solve it neither remember it. However, with the support of the LSP system, our CBP system can handle new problems easily by remembering the solutions provided by the LSP system. Therefore, our system does not have to be initialized with a huge case base. It has the potential to learn totally new experience in the runtime.

As a conclusion, integrating CBP to LSP is not only an idea that has the potential to improve the performance of pure LSP, but also a promising innovation to overcome some deficiencies of traditional CBP in the aspects of usage, efficiency and implementation.

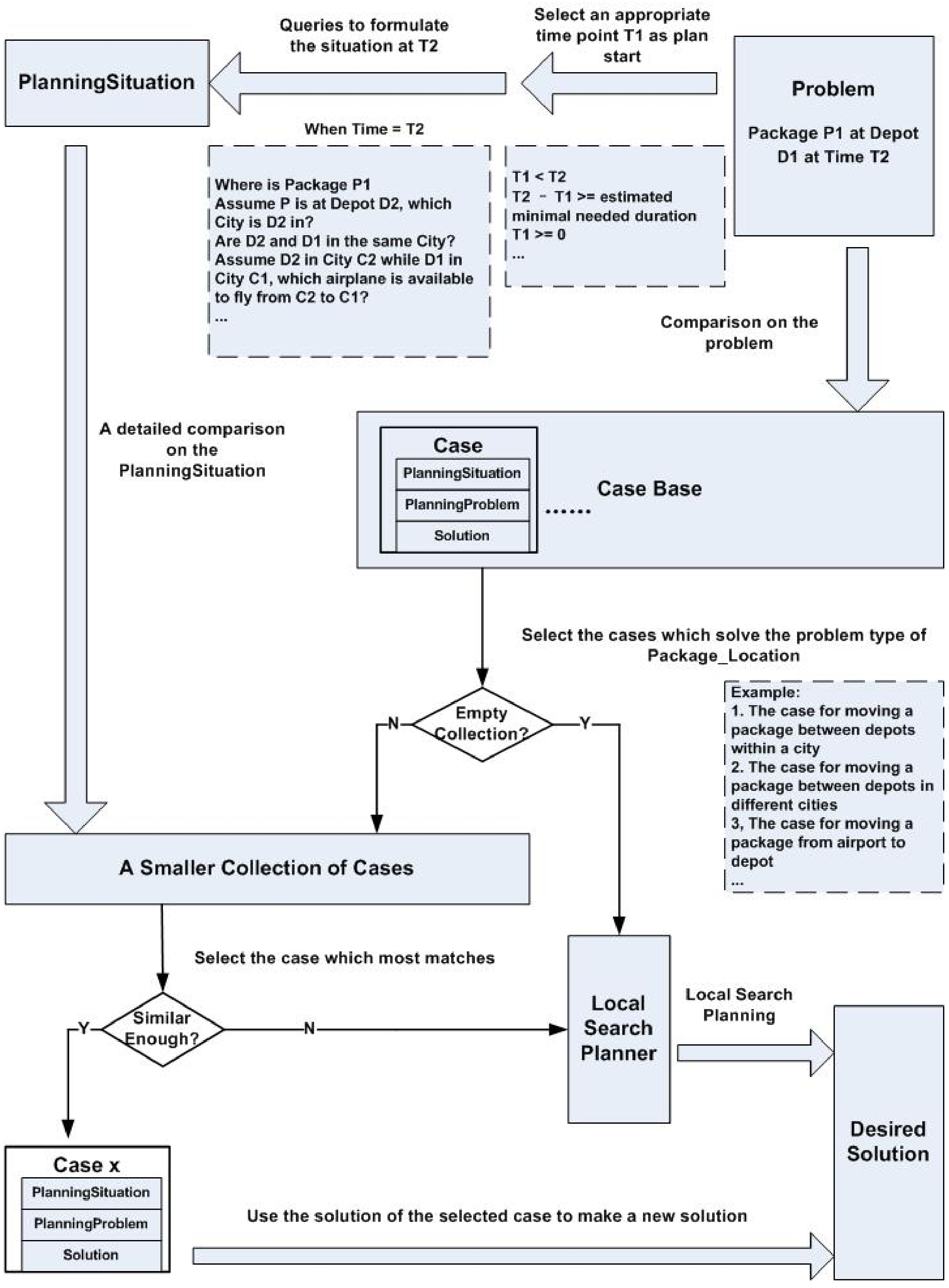

Queries to formulate the situation at T2
Select an appropriate time point T1 as plan start
PlanningSituation
Problem
Package P1 at Depot D1 at Time T2
When Time = T2
Where is Package P1
Assume P is at Depot D2, which City is D2 in?
Are D2 and D1 in the same City?
Assume D2 in City C2 while D1 in City C1, which airplane is available to fly from C2 to C1?
...
T1 < T2
T2 – T1 >= estimated minimal needed duration
T1 >= 0
...
Comparison on the problem
A detailed comparison on the PlanningSituation
Case
PlanningSituation
PlanningProblem
Solution
......
Case Base
Select the cases which solve the problem type of Package_Location
Empty Collection?
N
Y
Example:
1. The case for moving a package between depots within a city
2. The case for moving a package between depots in different cities
3, The case for moving a package from airport to depot
...
A Smaller Collection of Cases
Select the case which most matches
Similar Enough?
Y
N
Local Search Planner
Local Search Planning
Desired Solution
Case x
PlanningSituation
PlanningProblem
Solution
Use the solution of the selected case to make a new solution

Figure 17: An Example to Explain the Whole Work Flow of the Hybrid System

4. PROTOTYPE IMPLEMENTATION

With the idea that using case-based planning to assist local-search planning, the theoretical model of the hybrid planner is established. A prototype of this hybrid planner is implemented based on the existing local-search planning system Crackpot.

4.1 The Local-Search Planning System: Crackpot

Crackpot is designed to be a real-time, anytime, dynamic, domain-independent planning system in a complex world. Crackpot's real-time design means the planner is bound by time to return a plan, even if approximate. The anytime property means that the planner continually improves the plan if more time is available. Crackpot can dynamically adapt to external changes during its computation such as made by other agents in the world. Crackpot models complex symbolic and numeric attributes of objects, such as in the real world, and reasons about finite resources. Crackpot makes plan to execute actions with durations, and concurrent and synergistic actions. Although Crackpot is designed to be domain-independent, domain specific heuristics may be "plugged into" the system and will automatically be used for planning.

Crackpot uses local search, based on iterative repair, for planning. Iterative repair has been shown to be useful for dynamic, re-planning situations and reasoning about time and resources. In iterative repair, fully grounded plans are improved in a series of repair iterations. In each of repair iteration, a *Cost* in plan is selected, and then one out of a set of heuristics is applied to reduce the *Cost*. The application of a heuristic changes the plan to form a new plan.

In Crackpot, *Cost* is an inconsistency in the plan. A *Cost* can be:

- an unachieved goal condition,
- an unsatisfied condition of an action already in plan, or
- use of an actuator beyond its capacity.

Each *Cost* is associated with an integer *CostValue* which reflects its severity, e.g., an unachieved goal has a *CostValue* of 1000 while a normal unsatisfied condition has a *CostValue* of 200. Therefore, the overall cost value reflects the over plan quality: a lower overall cost value indicates less critical inconsistencies existing in the plan, which means a better plan quality and thus a better planning result.

The selected heuristic evaluates several options to select a few appropriate changes to the plan. Some types of change of plan are:

- Insert a new action to plan, e.g., in an attempt to repair an unachieved goal condition.
- Move an existing action in plan to some other start time, e.g., to appropriately schedule usage of an actuator.
- Remove an action from plan, e.g., remove an action with a difficult to satisfy condition.
- Change a parameter of an existing action in plan, e.g., to satisfy a condition of an action following in time in the plan.

The selected change is then applied to the existing plan to produce a new plan. Some details of the heuristics in the context of domain-independent planning can be found in [26].

Crackpot is an evolution of the system EXCALIBUR [26] adapted to planning. The Crackpot system is supposed to act as a platform the research project on Automated Story Generation for Games in an associated lab. Crackpot has been implemented as an open source software (see [27]), primarily in C++, with a public domain license. More details about the Crackpot can be found at [27].

4.2 The Hybrid Planning System

The hybrid planning system is the implementation of the developed hybrid planning approach. It integrates the case-based planner into the local-search planner Crackpot, and attempts to exploits both of their advantages. In this prototype, the case-based planner is first implemented and later the hybrid planning system is obtained from the integration.

4.2.1 Overall Design

As shown in Figure 17, in the hybrid planning system, the local-search planning system (Crackpot) is the main process. It iteratively repairs the plan to achieve the goals by optimizing its consistency. In Crackpot, the inconsistency of the plan is reflected by the calculated overall cost value. In each of iteration, the local-search planner selects one problem (represented by cost in Crackpot) as the target problem and chooses one appropriate heuristic to solve it. In the hybrid planner, the

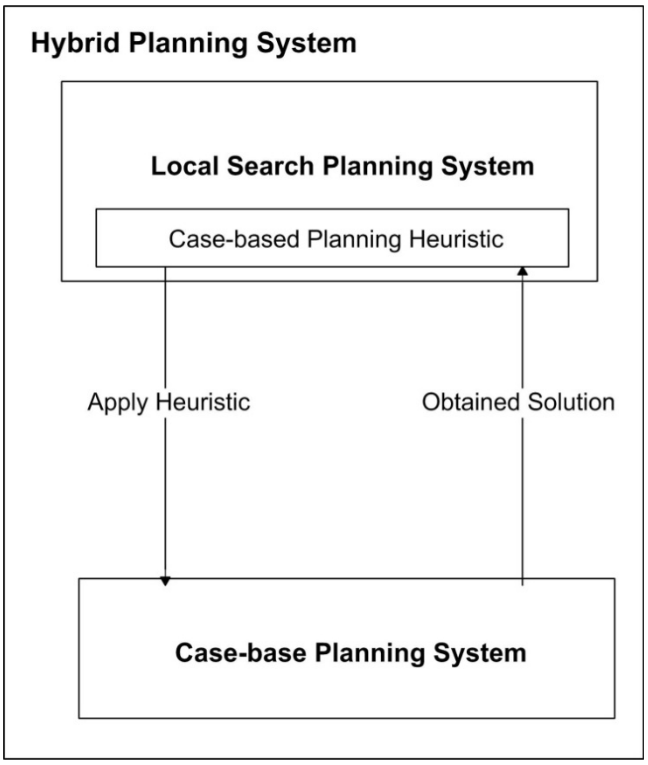

Hybrid Planning System
Local Search Planning System
Case-based Planning Heuristic
Apply Heuristic
Obtained Solution
Case-base Planning System

Figure 18: Overall Structure of the Hybrid Planning System

case-based planner is integrated into Crackpot through a new local-search heuristic called case-based planning (CBP) heuristic. The local-search planner can trigger the case-based planning system by calling the CBP heuristic for service. When the case-based planner completes its planning process, it will pass the obtained solution to the main process, which will apply the solution and realize the suggested changes.

4.2.2 Components

As the local-search planning system Crackpot is already working, the implementation involved in this research mainly focuses on the development of the case-based planning system and the integration work.

Inside the CBP system, there are four major components (see Figure 18) to carry out the case-based planning process: CasePlanner, CasePlanManager, CaseManager and CaseLibrary.

CasePlanner represents the whole CBP system and it includes all the other system components as its own members. It is also the top controller of the workflow and in charge of the activities including system initialization, planning processing, and the communication with the main system. It has the methods to receive external commands to start planning, collect and process data, produce solutions and send out solutions. Another important component inside CasePlanner is CaseDomainPackage, which maintains all the object parameters and value parameters used in cases.

CasePlanManager has two major responsibilities. Firstly it generates the *PlanningSituation* which describes the target problem and working situation. This is the task for case retrieve. The generated *PlanningSituation* is then passed to CaseManager to find out the best matching case. Secondly, CasePlanManager instantiates and optimizes the solution suggested by the selected best case. This is the task for case reuse. As mentioned before, before the solution added into the plan, these parameters need to be replaced with real instances that substantially exist in the planning world.

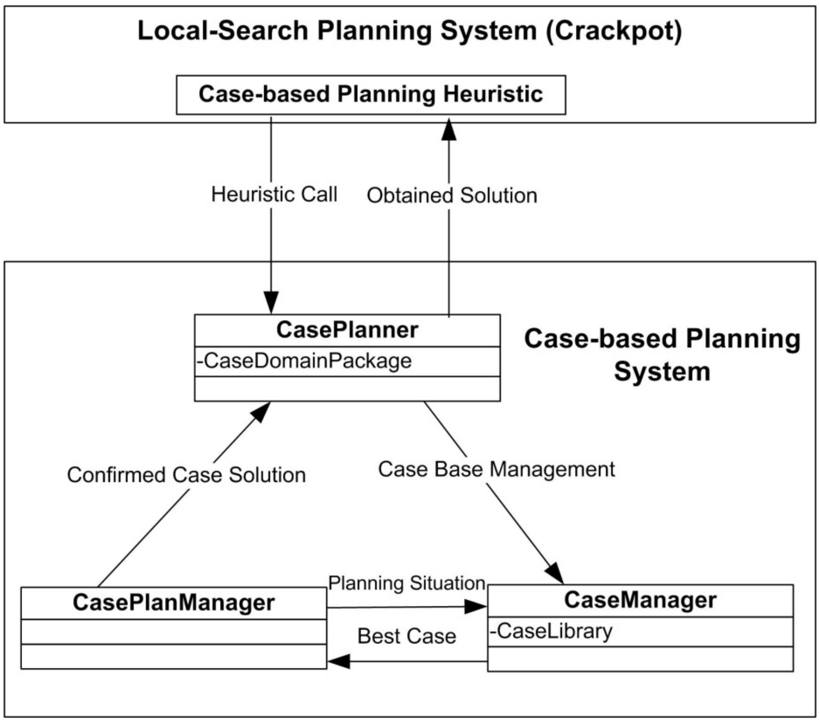


Figure 19: The Major Components and Workflow of the CBP System

CaseLibrary is the place where cases are stored in the runtime. It has the basic methods for case base management, including case insertion, case deletion and case modification. It is a component of CaseManager.

CaseManager is mainly in charge of case matching, which is a subtask of case retrieve. It receives the *PlanningSituation* from CasePlanManager, and searches for the best matching case from CaseLibrary. It also takes some other important responsibilities including guiding CaseLibrary to carry out case base management. This is the work for case revise and case retention.

The integration of these two planning systems is accomplished by defining a new local-search heuristic, the case-based planning heuristic. It is just an interface and shares the same structure with other local-search heuristics.

4.2.3 Implementation

The implementation work takes over 5000 lines of code in C++ and employs Visual Studio 2008 as the development tool. C++ standard library and boost library are utilized.

A detailed class diagram of the implementation is given in Figure 19.

This project is open-source with a public domain license and the source code repository is allocated on Souceforge.net.

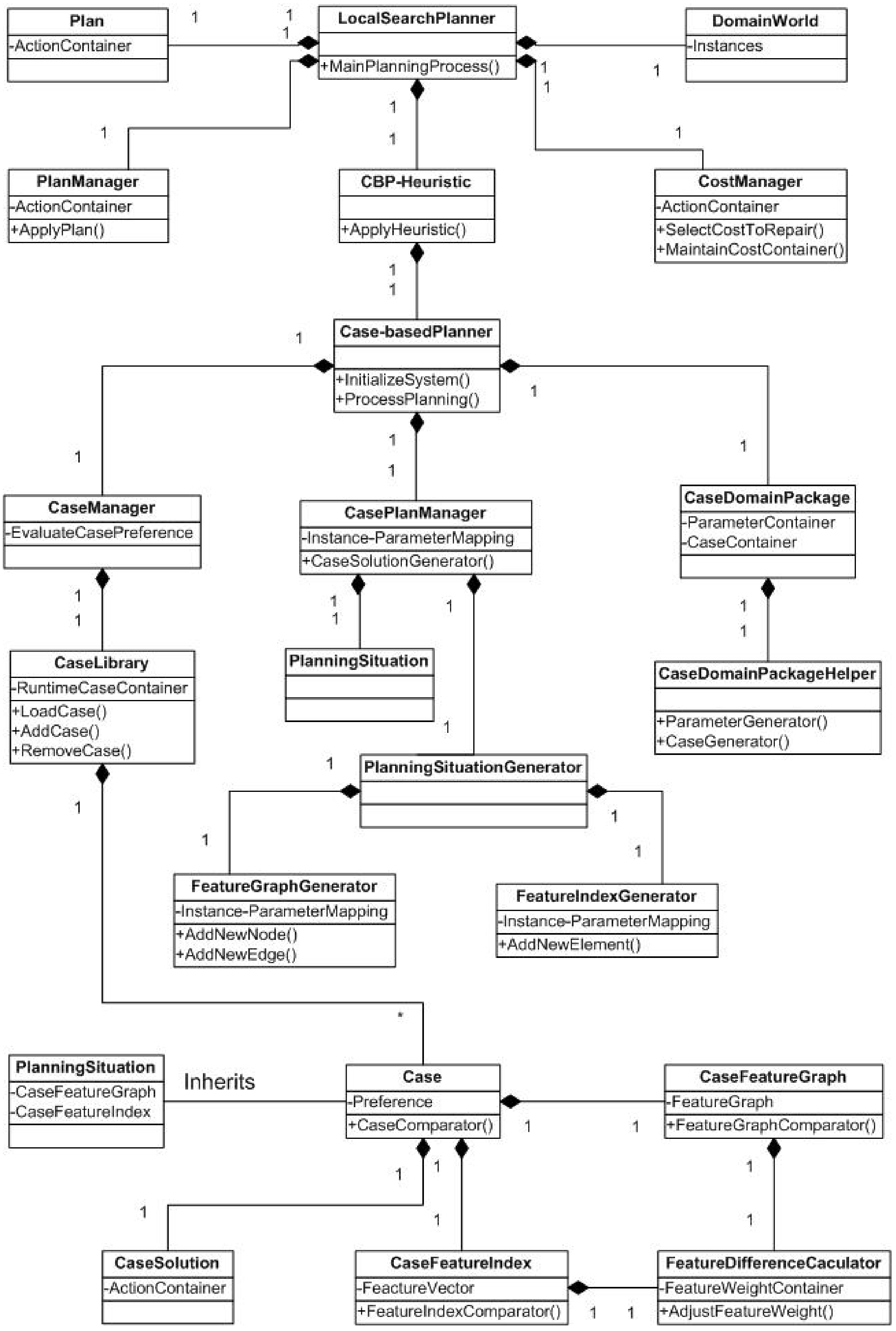


Figure 20: Class Diagram of the Implementation (Mainly the CBP System and the Integration Part)

5. EVALUATION

The prototype of the hybrid planning system is tested by referring a series of organised experiments. Its performance is evaluated by comparing to the performance of the original system. The planning problems in the experiments are constructed using classic logistics domain.

5.1 Sample Domain

Logistics application domain (see Figure 20) is a classic and benchmark application domain that is widely used for testing planning systems.

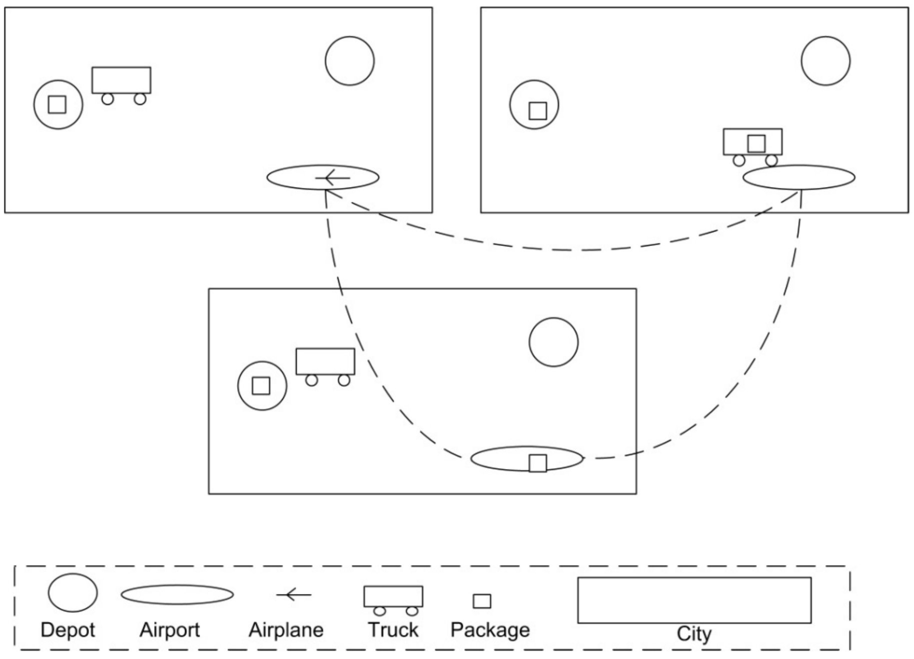


Figure 21: Logistics Domain

In problems of logistics domain, there are some basic objects including cities, depots, airports, airplanes, trucks, and packages. In each city, there are several depot(s) and airport(s). Trucks can move between locations which can be depots and airports within one city. Airplanes can fly only between airports across cities. Each truck and airplane can only carry a certain number of packages. Trucks, airplanes, depots and airports can be the places to allocate packages. Initially, the locations of all the objects are known. The goal in the logistics domain is to find a plan for delivering a number of packages to their destinations.

Some actions are provided in the domain and a plan is organized as a sequence of these actions. The actions defined in our planning problems are given in Table 1.

Action	**Description**	**Duration**
Load_Truck(*p*, *t*, *l*)	Load package *p* onto truck *t* at location *l*	10
Unload_Truck(*p*, *t*, *l*)	Load package *p* onto truck *t* at location *l*	10
Move_Truck(*t*, *l1*, *l2*)	Move truck *t* from location *l1* to location *l2*	20
Load_Airplane(*p*, *a*, *ap*)	Load package *p* onto airplane *a* at airport *ap*	20
Unload_Airplane(*p*, *a*, *ap*)	Load package *p* onto truck *t* at airport *ap*	20
Fly_Airplane(*a*, *ap1*, *ap2*)	Fly airplane *a* from airport *ap1* to airport *ap2*	100

Table 1: Actions in Logistics Domain

In logistics domain, the problem difficulty goes up when the number of involved objects increases. This advantage makes it convenient to create problems with different difficulty levels to construct the experiments. More analysis about logistics domain can be found in [34].

There are three test problems prepared in the experiments, one at easy level, one at intermediate level and one at hard level. Different level of problems involves different number of objects in the problem definition and requires delivering different number of packages in the goal specification. The detailed problem definitions are given in Table 2.

	Easy	Intermediate	Hard
Problem ID	1	2	3
City	2	7	14
Depot	2	7	14
Airport	2	7	14
Truck	2	7	14
Airplane	1	2	4
Package to Deliver	4	20	40

Table 2: Test Problems

In the current stage of experiment, the cases used to perform case-based planning are manually created. The detail of these cases is illustrated in Table 3.

No.	Case Problem Type	Cases and Descriptions
1	Package Location	Move package between depots inside a city
2		Move package between depots across cities
3		Move package from depot to airport across cities
4		Move package from airport to depot across cities
5		Move package from airport to airport across cities when there is no airplane
6		Move package from airport to airport across cities when there is an airplane
7	Truck Location	Move truck between depots inside a city

8	Airplane Location	Move airplane between airports across cities
9	Truck Capacity	Unload the less useful packages from a truck and load the useful one
10	Airplane Capacity	Unload the less useful packages from an airplane and load the useful one

Table 3: Cases Created for the Experiments

5.2 Experiment Configuration

Observed from some earlier experiments, the performance of the two systems varies across test runs. Sometimes, there are big differences. This is because of the use of some random parameters. Therefore, to obtain the average performance, the two systems are both configured to run 100 test runs for each planning problem. Moreover, the purpose of the experiments is to examine the planning efficiency of the two systems in both planning speed and results. Therefore, we will mainly collect the data like the time expense in planning and the overall cost value changing trend over time. (In Crackpot, a lower overall cost value reflects a better plan quality and a better planning result. See Section 4.1 for more details.) Besides, some other data, e.g., the heuristic usage frequency, is also recorded for further analysis.

As mentioned before, the local-search planner is the main process in the hybrid system and it works with iterative planning iterations. For test problems with different difficulty, there are different maximal numbers of planning iterations to be executed in one test run. These numbers are used to make the experiments efficient, avoiding spending time on useless efforts. But they first have to make sure of an enough period of time for either of the planning systems to produce its best performance in one test run. (In Crackpot, the best performance means reducing the overall cost from the initial value to the lowest value it can ever reach.) With these rules, the maximal iteration numbers for the different test problems are configured according to earlier observations.

To analyze the performance of these two systems, we not only have to know what the lowest cost values can ever be achieved, but also how much time the system takes to reach there. However, the lowest cost value varies very much across different problems, different systems and different test runs. Especially, neither of the two systems can constantly reduce the cost values to zero even for the simple level problem. Therefore, for a better comparison, different benchmark cost values are set up for different level of test problems. These values are selected based on the earlier observations in the previous experiments. They are relatively very low values, but they are also high enough that both planning systems can get there in most of the test runs. The time expense from the planning beginning to the moment when these benchmark cost values are reached is to be recorded for analysis. (As a result, these benchmark cost values turn out to be nearly 1% of the respective initial cost values.) The selected benchmark cost values are given in Table 4.

	Easy	Intermediate	Hard
Initial Cost Value	3000	20000	40000
Max Iteration Number	200	400	600
Cost Benchmark	20	200	500
Benchmark / Initial Cost	0.67%	1%	1.25%

Table 4: Benchmark Cost Values of Different Test Problems

All the experiments are performed with the computer configuration as: Intel Core2 Quad CPU Q9550@2.83 2.83GHz, 4GB memory, 32-bit Windows Vista operating system.

5.3 Performance Observations

With the obtained experimental results, a series of analysis have been taken from different aspects and several important observations are obtained.

5.3.1 Experiment 1: Easy Level Problem

This section introduces the performances of the two planning systems in solving the easy level logistics problem (problem 1).

Overall Performance

The overall performance of the original system and hybrid system are shown in Figure 21 and Figure 22 respectively. As they are both three dimensional graphs, they are displayed from four different angles for a better view. The situation is the same for Figure 25, 26, 29 and 30.

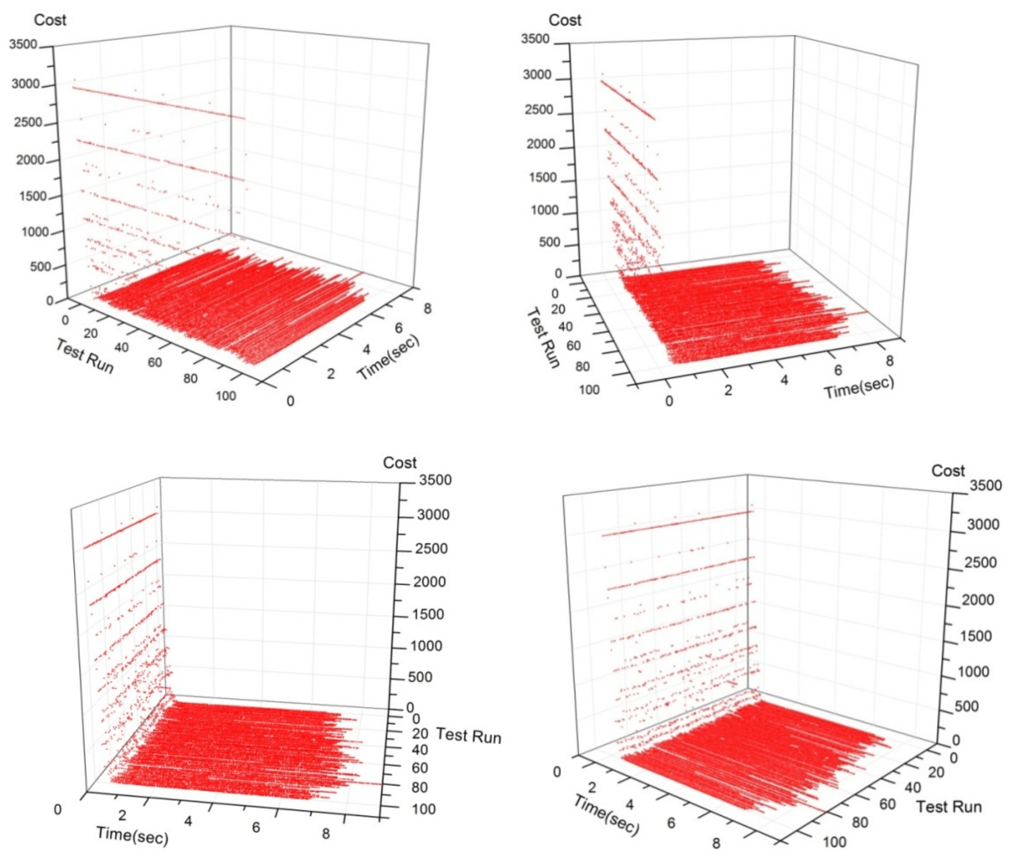
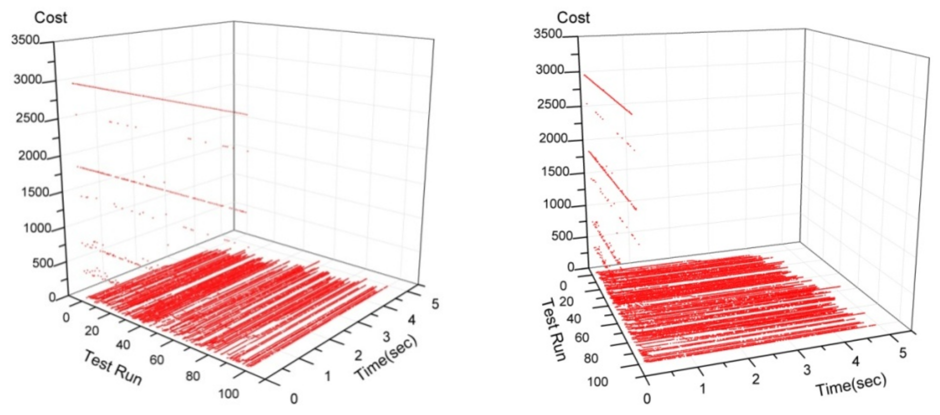


Figure 22: Overall Performance of the Original System in Experiment 1



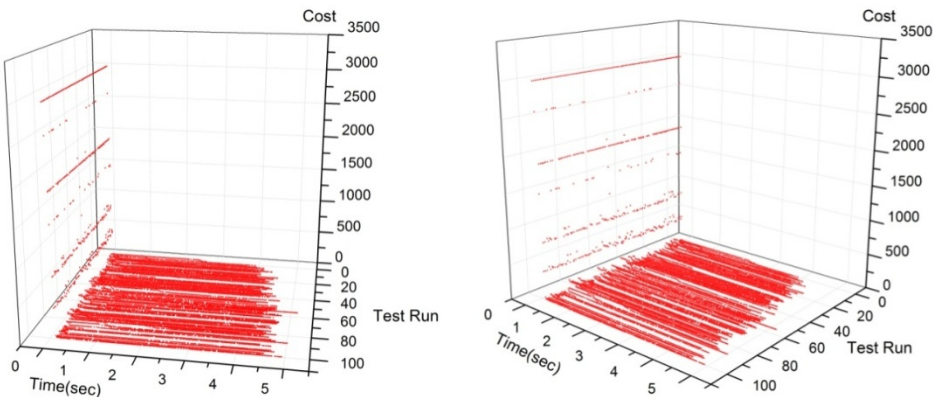


Figure 23: Overall Performance of the Hybrid System in Experiment 1

Figure 22 and 23 (and the similar figures in the following section) present how the cost value changes over time in all test runs in the experiment. They are to provide an overall view of the two systems' performances. In these figures, data can be read and analyzed by test runs. In every test run, the cost is reduced from the initial value to the lowest value and a cost-time dot curve is drawn accordingly. In every test run, we examine the obtained dot curve and find out the first point which has the lowest cost value. Then the time expense that the system uses to make the best performance in that particular test run can be roughly calculated. From the last point of each cost-time dot curve, the total time expense in that particular test run for finishing the given number of planning iterations can be roughly read.

Figure 22 shows that, in one test run, the original system uses roughly 0.55 seconds in average to achieve the lowest cost value, and around 7.20 seconds in average to finish all 200 planning iterations. In contrast, Figure 23 shows that, the hybrid system uses roughly 0.25 seconds in average to achieve the lowest cost value, and 4.45 seconds in average to finish all 200 planning iterations.

Planning Speed

The planning speed of the two systems is evaluated by measuring the time expense taken to reach the benchmark cost value (20).

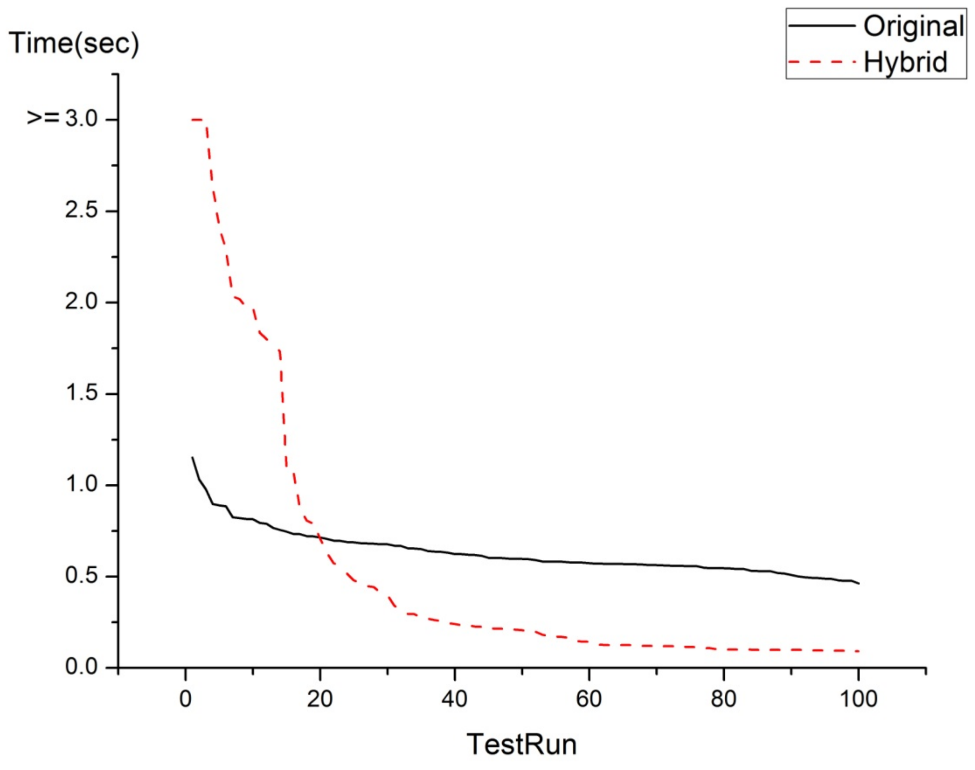


Figure 24: Time Expense in Achieving the Cost of 20 in Experiment 1 (Test Runs Sorted According to Time)

In Figure 24, the values are sorted according to the time expense in descending order. This is only to provide a clearer view of the comparison, not saying the time expense always reduces when test run goes. This also applies to the similar figures in the following. From Figure 24, firstly, the data reflects that the hybrid system fails to achieve the benchmark cost value in 2 test runs, while the original system can always reach there. Moreover, for most of the test runs, the hybrid system plans clearly faster. To achieve the benchmark cost value, the hybrid system needs to spend around 0.35 seconds in average, while the original system needs to take around 0.65 seconds in average. The ratio is roughly 1: 1.8.

Planning Results

The planning results of the two systems are evaluated by measuring the lowest overall cost values they have ever achieved in the test runs. A lower overall cost value indicates less critical inconsistencies existing in the plan, which implies a better plan quality and thus a better planning result.

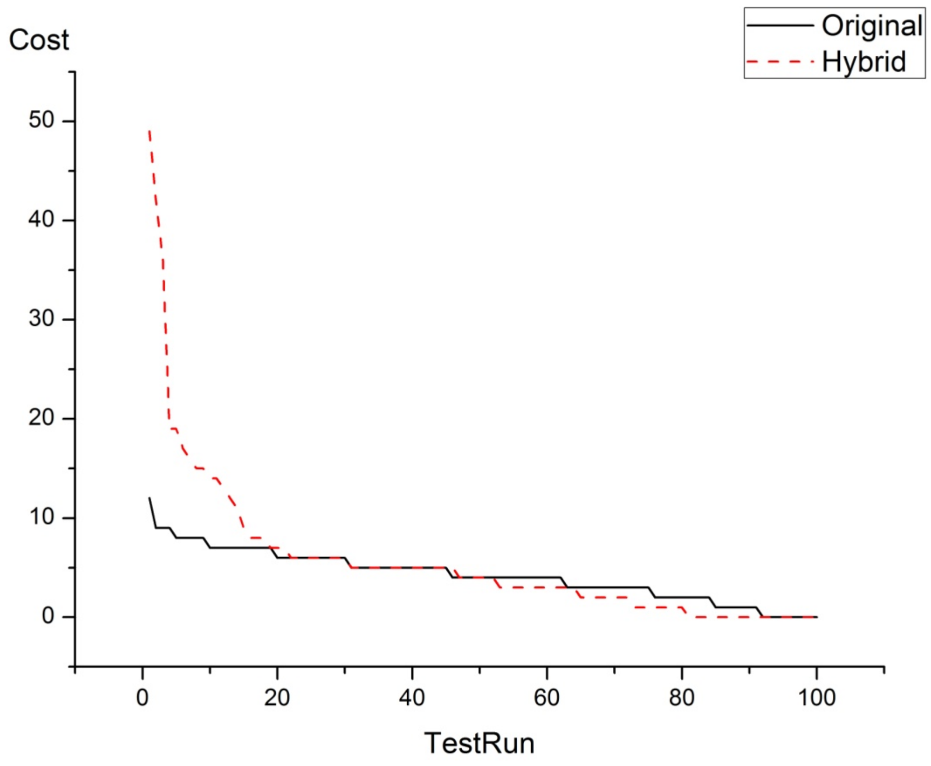


Figure 25: The Lowest Cost Values Achieved in Experiment 1 (Test Runs Sorted According to Cost)

In Figure 25, the values are sorted according to the lowest cost values that a test run ever achieves in descending order. This is only to provide a clearer view of the comparison, not saying the lowest cost values always reduces when test run goes. This also applies to the similar figures in the following. As shown in Figure 25, the average lowest cost values that these two systems can normally achieve in a test run are almost the same: about 5 for the original system and 7 for the hybrid system. However, the hybrid system has some possibility to fail to reduce the cost value below 20, while the original system has its lowest cost values below 10 in every test run. Nevertheless, there are 16 test runs in which the hybrid system successfully reduces the cost to zero, while that number for the original system is only 8.

In conclusion, in experiment 1, which is based on the easy level problem, the hybrid system performs much faster but slightly less stable. The lowest cost values they can achieve are similar, which reflect their similar performance in plan quality and planning results.

5.3.2 Experiment 2: Intermediate Level Problem

This section introduces the performances of the two planning systems in solving the intermediate level logistics problem (problem 2).

Overall Performance

The overall performance of the original system and hybrid system are reflected in Figure 26 and Figure 27 respectively.

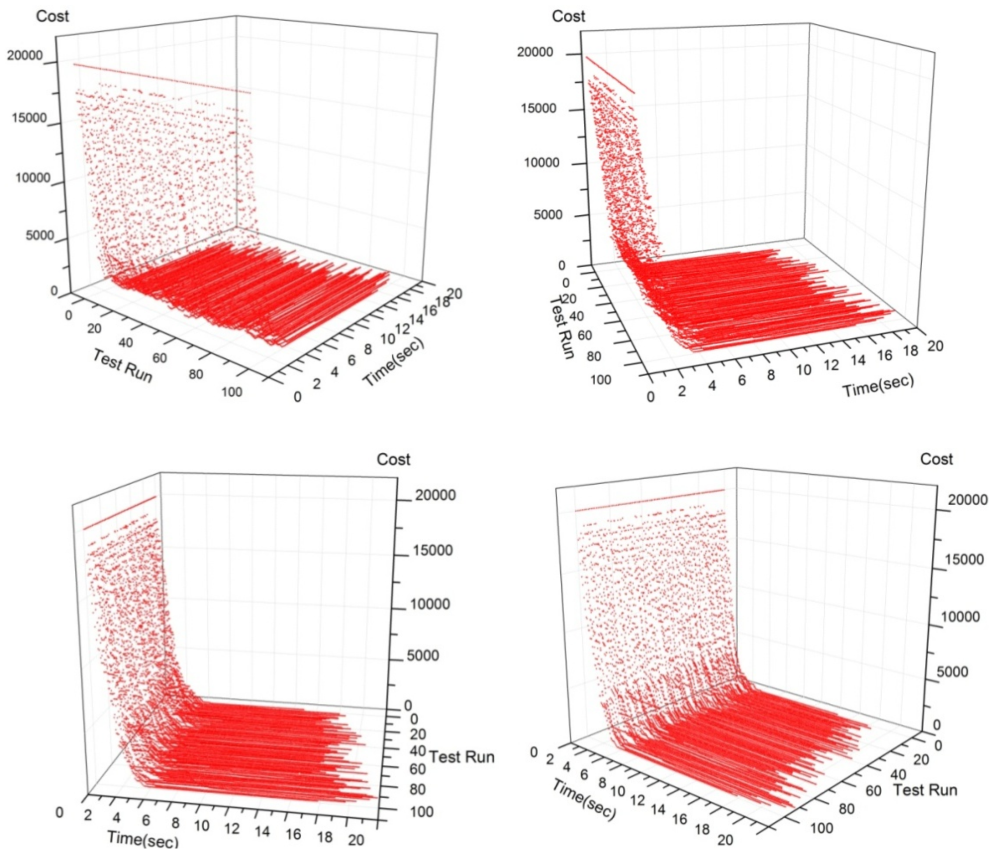
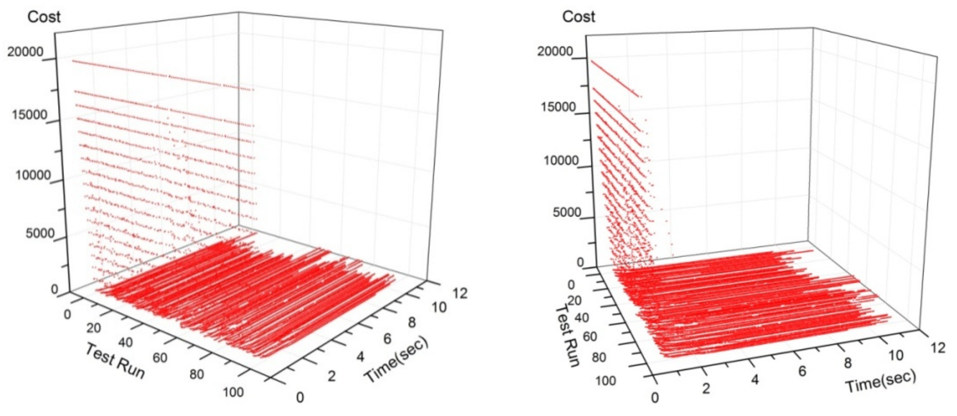


Figure 26: Overall Performance of the Original system in Experiment 2



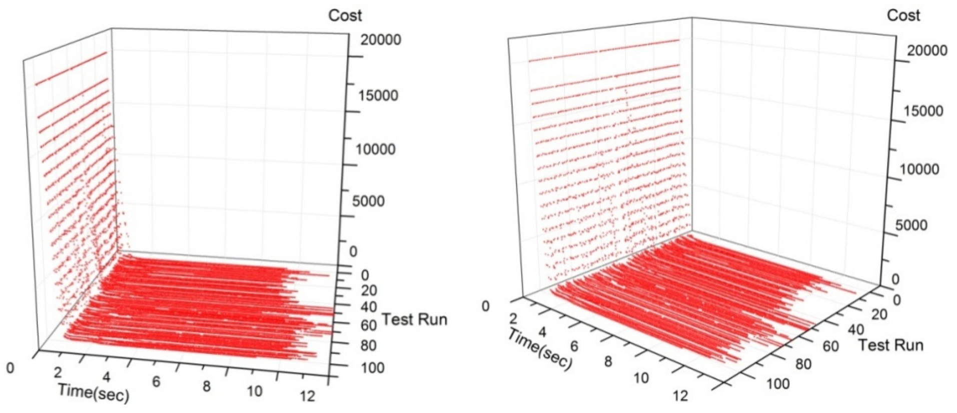


Figure 27: Overall Performance of the Hybrid System in Experiment 2

Figure 26 indicates that the original system uses roughly 3.65 seconds in average to achieve the lowest cost value, and around 16.50 seconds in average to finish all 400 planning iterations. In contrast, Figure 27 shows that the hybrid system takes around 0.95 seconds in average to achieve the lowest cost value, and about 9.80 seconds in average to finish all 400 planning iterations.

Planning Speed

The planning speed of the two systems is evaluated by measuring the time expense taken to achieve the benchmark cost value (200).

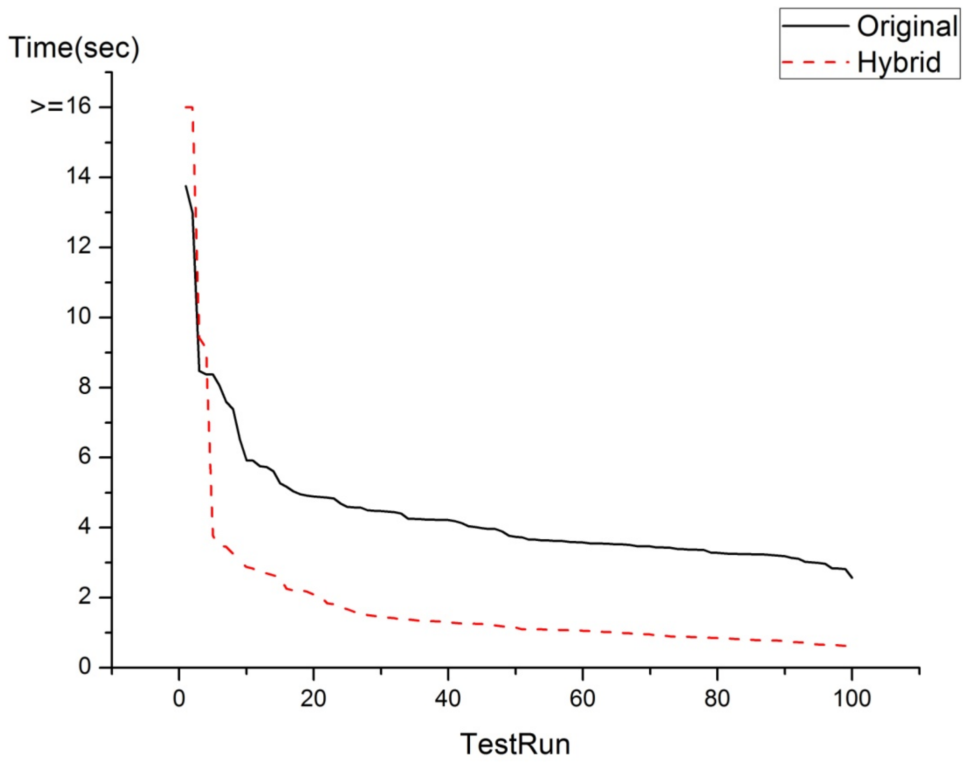


Figure 28: Time Expense in Achieving the Cost of 200 in Experiment 2 (Test Runs sorted According to Time)

As shown in Figure 28, in experiment 2, the hybrid system again has two test runs that fail to reach the benchmark cost value, while the original system still keeps a reliable performance on this aspect. However, in most of the test runs, the planning speed of the hybrid system is clearly much faster. To achieve the benchmark cost value, the average time expense for the hybrid system is around 1.25 seconds, while it is around 4.05 seconds for the original system. The ratio is about 1: 3.2.

Planning Results

The planning results of the two systems are evaluated by measuring the lowest overall cost values they have ever achieved in the test runs.

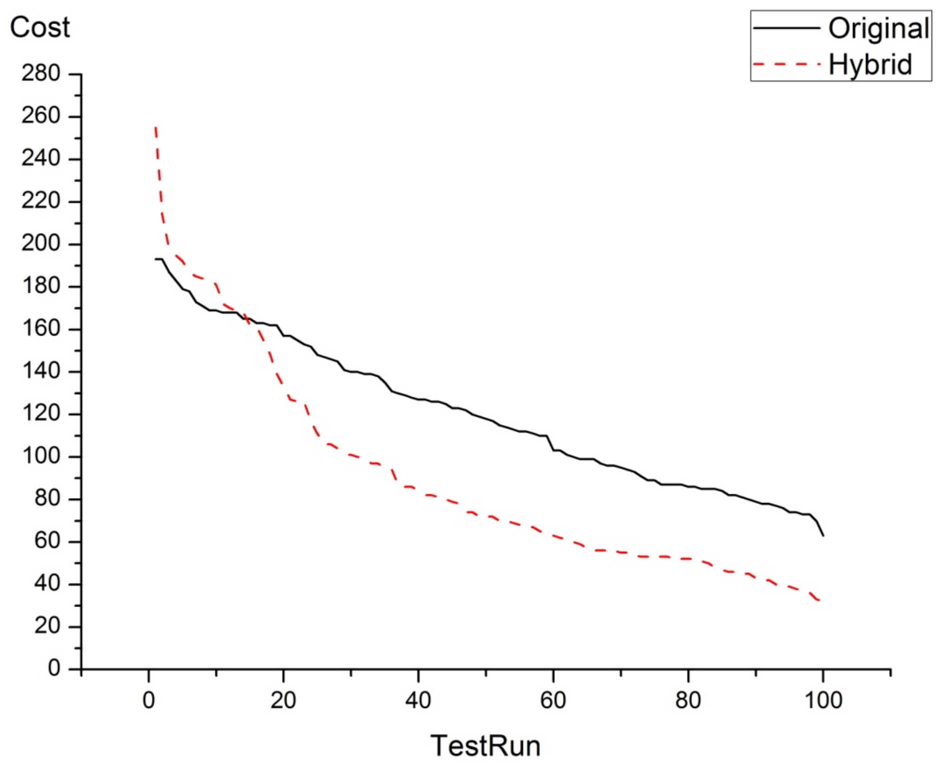


Figure 29: The Lowest Cost Values Achieved in Experiment 2 (Test Runs Sorted According Cost)

Figure 29 clearly indicates the planning results of the hybrid system are better than the original system. The average lowest cost value that the hybrid system can normally achieve is around 90, while the number is around 120 for the original system. More importantly, the hybrid system shows its capability to reduce the cost value to very low levels that the original system can never make. As Figure 29 shows, the hybrid system reaches a lowest cost value below 70 in 1/3 of its test runs, and achieves 40 as its best performance. In contrast, the original system never reaches a cost value below 70. However, the hybrid system has bad performances in two test runs, stopping at the cost value above 200, while the original system has no such failures and performs more reliable and stable.

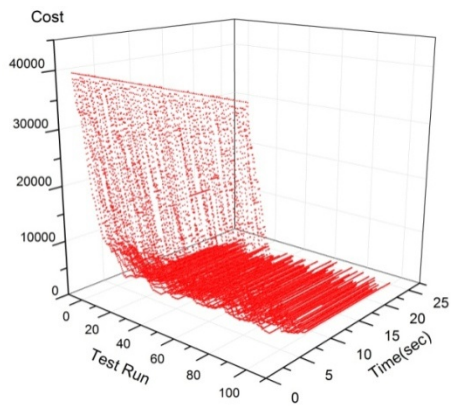
Evaluating the experimental results for solving problem 2, the conclusion is similar to which was obtained in experiment 1. The hybrid system demonstrates an even better planning speed comparing to the original system, but it is still slightly less stable. Moreover, in this round, the hybrid system has obtained better planning results by reducing the costs to much lower values.

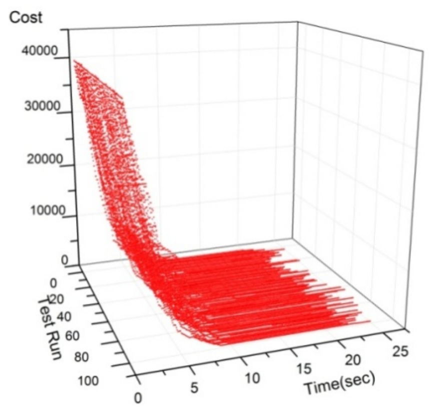
5.3.3 Experiment 3: Hard Level Problem

This section introduces the performances of the two planning systems in solving the hard level logistics problem (problem 3).

Overall Performance

The overall performance of the original system and hybrid system are shown in Figure 230 and Figure 31 respectively.





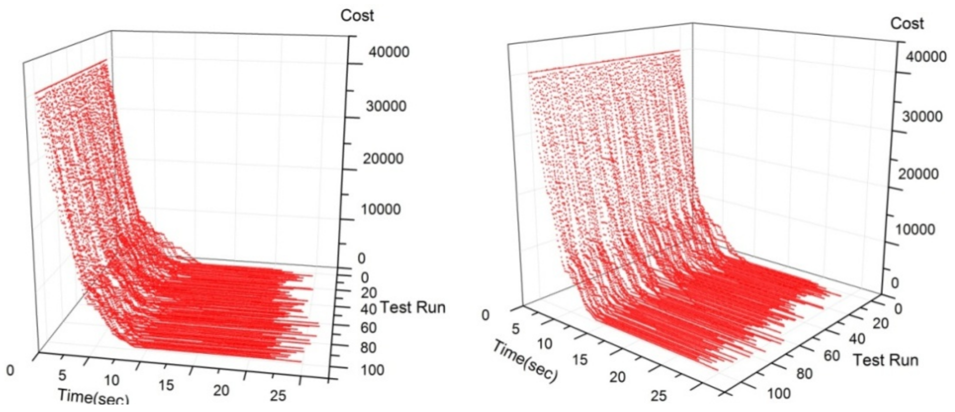


Figure 30: Overall Performance of the Original System in Experiment 3

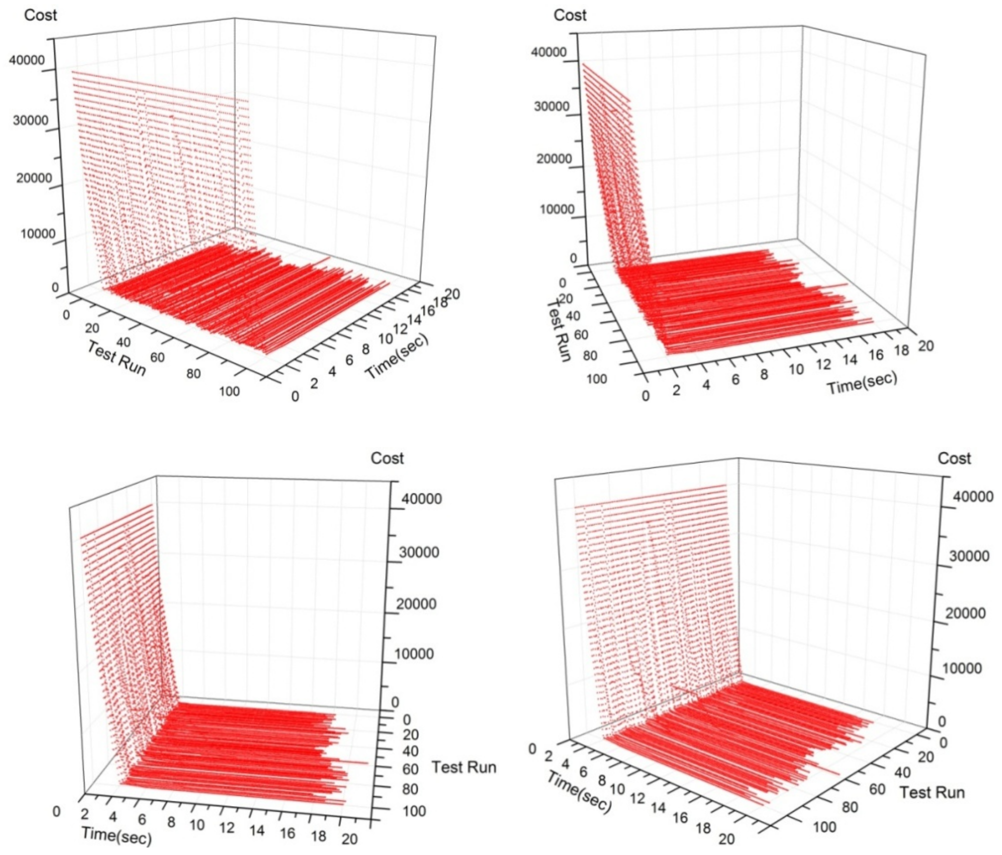


Figure 31: Overall Performance of the Hybrid System in Experiment 3

Figure 30 indicates that the original system uses about 9.85 seconds in average to achieve the lowest cost value, and around 22.50 seconds in average to finish all 600 planning iterations. Figure 31 shows that the hybrid system takes around 2.40 seconds in average to achieve its lowest cost value, and about 16.80 seconds in average to finish all 600 planning iterations. Furthermore, in this experiment, the lowest cost values that the two systems can ever achieve differ a lot.

Planning Speed

The planning speed of the two systems is evaluated by measuring the time expense taken to achieve the benchmark cost value (500).

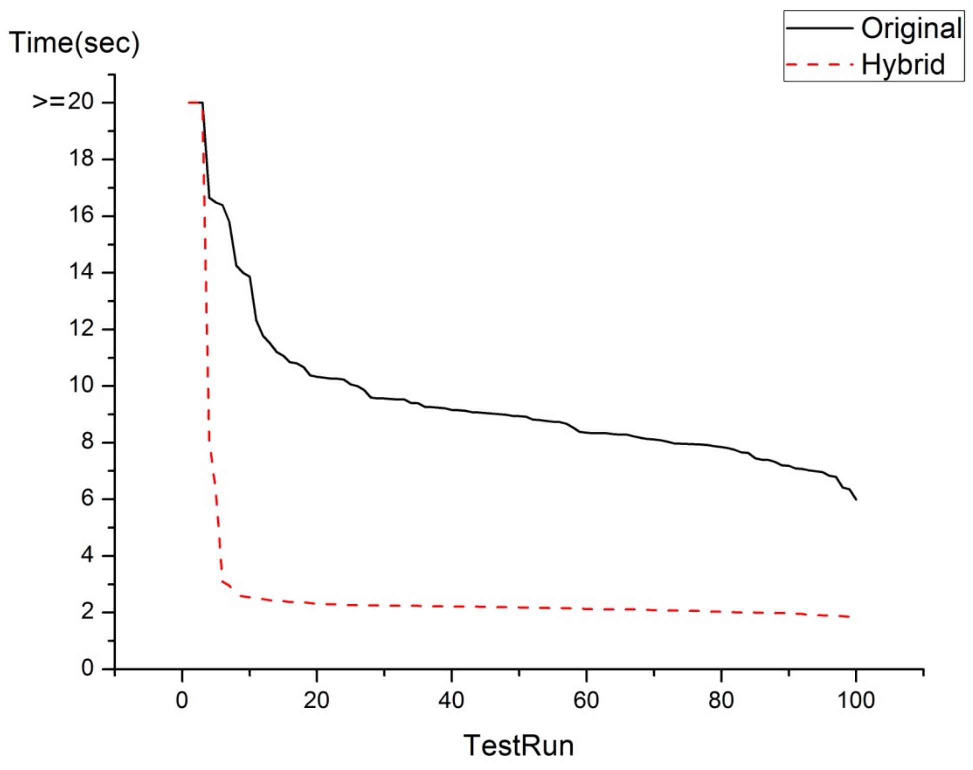


Figure 32: Time Expense in Achieving the Cost of 500 in Experiment 3 (Test Runs Sorted According to Time)

As shown in Figure 32, in experiment 3, the hybrid system has 3 test runs that fail to reach the benchmark cost value. But this time, the original system also has such failures in 3 test runs. Their probabilities of failure are the same. Moreover, in most of the test runs, the hybrid system still shows a much faster planning speed. To achieve the benchmark cost value, the average time expense for the hybrid system is around 2.10 seconds, while it is around 8.80 seconds for the original system. The ratio is about 1: 4.2

Planning Results

The planning results of the two systems are evaluated by measuring the lowest overall cost values they have ever achieved in the test runs.

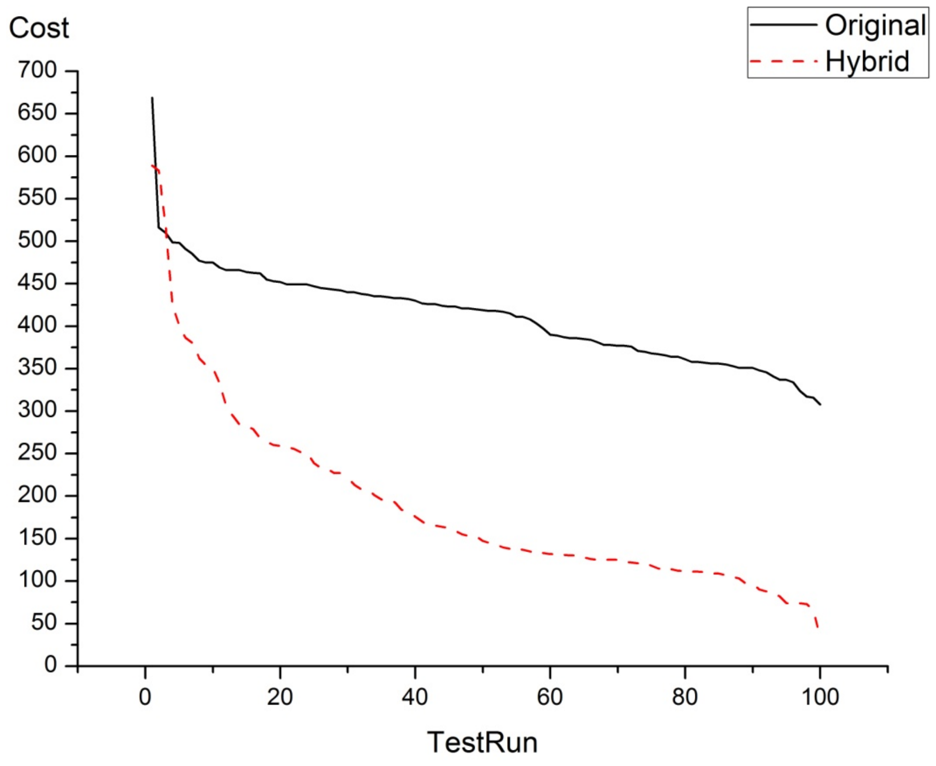


Figure 33: The Lowest Cost Values Achieved in Experiment 3 (Test Runs Sorted According to Cost)

Figure 33 indicates that, the planning results of the hybrid system are much better than the original system. The average lowest cost value of the hybrid system can normally achieve is around 160, while that number is around 420 for the original system. Similar to the results in experiment 2, the hybrid system again performs its capability to make a better plan than the original system. In Figure 33, the hybrid system reduces the cost below 270 in most of the test runs, and reaches 36 as the best performance, while all that results of the original system are above 300. Moreover, the hybrid system still has 3 bad performances, stopping at the cost value above 500. But in this time, the original system also has 3 such failures.

Evaluating the performances of the two systems in experiment 3, the hybrid system still wins out in planning speed the same as in all previous experiments. Moreover, the hybrid system again obtains much better planning results than the original system, and the stability of the two systems is almost the same in this experiment.

5.4 Analysis

From the experimental results, we make some observations and analysis obtained from various aspects.

5.4.1 Performance Analysis

In all three experiments, the hybrid system demonstrates a much better performance in both planning speed and planning results. With analysis, these improvements are potentially achieved due to the following factors.

The CBP heuristic is considerably fast and efficient comparing to other regular local-search heuristics. According to the statistics, a single planning iteration that applies the CBP heuristic takes around 0.02 seconds in average. In contrast, the regular add-single-action requires around 0.02 seconds per iteration and the regular remove-single-action requires around 0.01 seconds per iteration. Especially, in most CBP planning iterations, there are multiple actions added into the plan as a whole while a regular local-search heuristic only adds in maximally one action in one iteration. Another observation is that, by bringing in the CBP heuristic, some regular local-search heuristics become less frequently used, e.g., add-single-action. As a result, the average time expense for a single planning iteration is reduced and the overall planning process is thus speed up.

The case-based planner plans with more intelligent and careful operations than other regular local-search heuristics. While planning, the case-based planner has a good capability to precisely identify the target problem and working situation at that moment, find out the most matching case from the case base, and eventually generate a proper solution for the target problem. As a sequence, the solutions provided by the CBP heuristic turn out to be desirable in most of the time. These high quality solutions contribute to a better plan quality and better planning result.

5.4.2 The Case-based Planning System

Based on the experimental results, the efficient and effective performance of the case-based planning system successfully improves the overall performance of the hybrid planning system.

With observations, the efficiency and effectiveness of the CBP system are potentially contributed by the following factors.

1. Effective information extraction. To solve a problem, the case-based planner needs to identify that problem and the working situation by extracting useful information. The information extraction technique employed by the case-based planner works well in the experiments. It always successfully obtained the correct and useful information with little time expense.
2. Fast and accurate case retrieving process. Case retrieve is recognized as the most computationally expensive task in the CBP process model because of the complex case matching process with plenty of cases. However, such complexity is controlled well in this case-based planner and the phase of case retrieve performs relatively fast and efficient. This is due to the well designed case retrieve method and case feature representation. Firstly, vector-based *CaseFeatureIndex* guides the matching process to the right category of cases, reducing the number of case candidates to be compared. Secondly, structure-based *CaseFeatureGraph* is managed to contain only the minimal number of nodes and edges which are the most needed. A small graph size naturally reduces the complexity of graph matching. More importantly, even with the minimal amount of information, the employed case matching technique is still able to make accurate matching in most of the time. According to the statistics, the case matching accuracy is higher than 98%. However, even with a low probability to fail, case mismatching still happens and it can cause serious planning mistakes that are not easy to be recovered.

Once case mismatching happens and the mismatched case is reused, the hybrid system is likely to obtain a bad performance in that test run.

3. Careful case reuse. Case reuse adapts the solutions suggested by the selected case to form the new solution to the current problem. Current case-based planner only adds actions into the plan and the number of actions is normally more than three. Adding more actions in one time is likely to improve the plan with a bigger step. But on the other hand, it is also likely to bring in more inconsistencies into the plan with that operation. Having been aware of this, there are two rules designed to make a careful case reuse. Firstly, if the case selected from caser retrieve is not similar enough, the case-based planner will give up that case and restart another working cycle to find a better one. If the second attempt still fails, the case-based planner will return an empty solution to avoid adding improper actions. Secondly, in the phase of case reuse, if some parts of the suggested solution cannot be correctly instantiated, that part is typically not needed for the current problem and will not be added in. In the experiments, these two rules are observed to be very helpful in eliminating incorrect case reuse.

5.4.3 Interaction of CBP and Local-search Planning

With experimental results, there are some other observations that indicate how case-based planning and local-search planning interact to each other in practise.

Firstly, local-search planning as the main process provides a good stage for case-based planning. In the hybrid planning system, the case-based planner is modelled as a local heuristic in a most applicable and straightforward way. The heuristic calling records indicate, in the beginning of planning, the local-search planner offers the same working chance to all heuristics, including the CBP heuristic. The CBP heuristic always gradually wins a higher chance to be selected with its good and stable performances. According to the statistics, the usage frequency of CBP heuristic in experiment 1, 2, 3 is 24.5%, 33.3% and 41.7% respectively. This integration method turns out to be workable in practise.

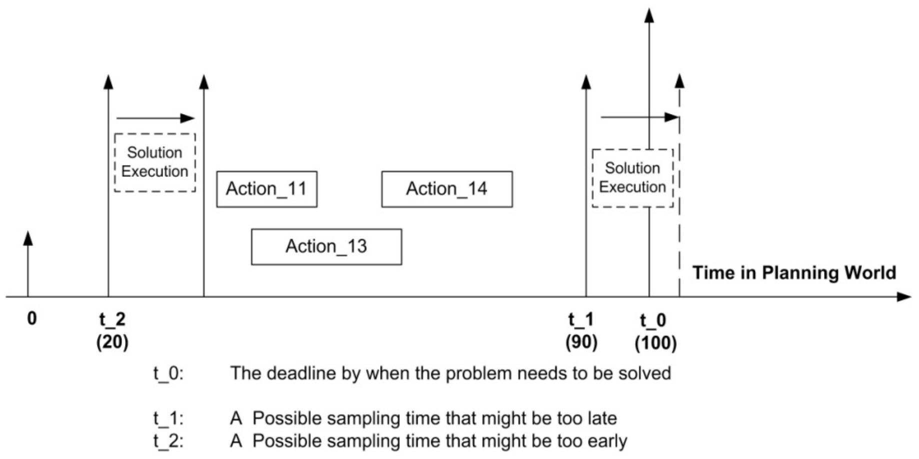
Secondly, these two planners assist to each other. As discussed before, with case-based planner, the hybrid planning system has a much better performance both in planning speed and planning results. On the other hand, the local-search planner provides the required information to the case-based planner to accomplish its planning process, and assists to repair its failures in the phase of case revise. When the case-based planner returns an empty solution or even a bad solution, the main process reduces the preference of the CBP heuristic, and calls some other local-search heuristics to solve that particular problem and repair the failures caused by the case-based planner. As illustrated before, if a mismatched case is reused by mistake, the improperly added actions are likely to bring in large inconsistencies into the plan and mislead the overall planning process. In many times, this kind of mistakes is corrected by the local-search planner by removing, moving or modifying the unexpected actions.

5.4.4 Limitations

In the experiments, the hybrid system does have several bad performances in some test runs, failing to reach the benchmark cost values. This is potentially due to the following limitations.

1. The case matching is not perfectly accurate. The mismatching mistake sometimes happens. If that mismatched case is accepted and reused, multiple unexpected actions will be added into the plan and bring in large inconsistencies which cannot be fully corrected easily. These inconsistencies sometimes will lead the planning into a bad direction. This is the major reason that why the hybrid system gets a higher chance to have bad performances in experiment 1 and 2.
2. It is not intelligent enough to determine the sampling time with random selection. In the current case-based planner, the sampling time, based on which to collect information to form the working situation, is randomly determined. The only rule for this selection is that the sampling time should be prior to the deadline by when the problem needs to be solved. This design shows its weakness in the tests. Sometimes, a sampling time is too late, leaving insufficient time to execute a solution. E.g., in Figure 34, a problem needs to be solved at time 100, while the sampling time is chosen at 90. If the

most matching case requires a least duration of 20 to execute its solution, this case is very likely to be given up before a detailed matching starts. This is another reason explains why a less similar case can be chosen without case mismatching.



Solution Execution
Action_11
Action_14
Action_13
Solution Execution
Time in Planning World
0
t_2 (20)
t_1 (90)
t_0 (100)
t_0: The deadline by when the problem needs to be solved
t_1: A Possible sampling time that might be too late
t_2: A Possible sampling time that might be too early

Figure 34: An Example of Bad Sampling Times

Sometimes a sampling time can be too early. E.g., in Figure 34, when truck_1 needs to be at depot_1 at time 100, the sampling time might be chosen at time 20. Even a right case is selected and a good solution moves the car to depot_1 by time 40, some other actions taking effect during time_40 and time_100 might change the location of truck_1 to another place. As a result, truck_1's location fails to be at depot_1 at time 100. This explains why the hybrid system sometimes has a bad performance even when the phase of case retrieve works appropriately.

3. Dependence of the case base. The current cases stored in the case base are constructed manually based on the earlier experimental results which were obtained by running the pure local-search planning system. Some

sub-problems appeared in those experiments were modelled as case problems and the obtained plans for these sub-problems were modelled as case solutions. For example, when plan with logistics domain, there are always requests to move a package from one depot to another. Therefore, it makes sense to create a case for this kind of problem. For that purpose, this sub-problem was tested on Crackpot. When Crackpot solves it, the problem itself and the obtained solution were used to formulate a case.

Observations also indicate, the performance of the hybrid planner very much depends on the quality of cases as well as the sample size of the case base. Firstly, case quality is very important. Each case must be valid and contain enough information to be sufficiently distinct. Sometimes even a single wrongly applied case can cause bad mistakes in planning which are hard to be repaired. For example, if a problem, move_package_inside_city, applies the case move_package_across_city, the planner can only correct this mistake by withdrawing all added actions or adding new actions to move the package back to the desired place, both of which require large efforts. Most of the failed test runs in the current experiments are caused by applying wrong cases. Secondly, the performance of the planner generally goes up when the size of case base increases. This is the observation I obtained in the process of implementing the case base. The more the cases are added in, the performance generally gets better. This finding is only empirical not theoretical. It seems imply that, with our current configuration of experiments, bringing in new cases has somehow reduced the overall complexity of planning even though the effort of case matching is going up.

5.5 Summary

Based on the experimental results, observations and analysis, a summary is discussed in this section.

According to the experimental results (see Table 5), the hybrid system demonstrates a much faster planning speed in all three experiments, and obtains better planning results in solving problem 2 and problem 3. As the problem difficulty increases, the performance difference between the two systems enlarges in both planning speed

		Original System	Hybrid System
Problem 1 (Easy Level)	Average Time to Achieve Benchmark Cost Value	0.65 sec	0.35 sec
	Average Time to Finish All Planning Iterations	7.20	4.45
	Average Lowest Cost Value	5	7
	Lowest Cost Value Ever Reach in All Test Runs	0	0
	Failure to Achieve Benchmark Cost Value	0	2
	Average Usage Frequency of CBP Heuristic	0	24.5%
Problem 2 (Intermediate Level)	Average Time to Achieve Benchmark Cost Value	4.05	1.25
	Average Time to Finish All Planning Iterations	16.50 sec	9.80 sec
	Average Lowest Cost Value	120	90
	Lowest Cost Value Ever Reach	72	40

	Failure to Achieve Benchmark Cost Value	0	2
	Average Usage Frequency of CBP Heuristic	0	33.3%
Problem 3 (Hard Level)	Average Time to Achieve Benchmark Cost Value	8.80 sec	2.10 sec
	Average Time to Finish All Planning Iterations	22.50 sec	16.80 sec
	Average Lowest Cost Value	160	420
	Lowest Cost Value Ever Reach	315	36
	Failure to Achieve Benchmark Cost Value	3	3
	Average Usage Frequency of CBP Heuristic	0	41.7%

Table 5: Summary Table for the Experimental Results

and planning results. In general, the hybrid system makes remarkable improvements on the original system and fulfils the research goals.

In addition, inside the hybrid planner, the case-based planner and the local-search planner work together closely and effectively in all three experiments. The local-search planner serves as the main planning process and assists the case-based planner to perform the task of case revise.

Moreover, in experiment 1, 24.5% of the planning iterations in one test run apply CBP heuristic, while that number is 33.3% in experiment 2 and 41.7% in experiment 3. As discussed before, the usage frequency of a heuristic is determined by its performances in the planning and it reflects the reliability of that heuristic. This statistics reveals that, the more difficult the problem is, the more frequently the CBP heuristic is applied. With this discovery, on the one hand, the case-based planner shows its important role inside the hybrid planner, on the other hand, the case-based planner demonstrates its better capability of handling complex planning problems in the experiments.

Simultaneously, there are also some limitations discovered in the current CBP and hybrid system, e.g., the sampling time is not so intelligent and the case matching is not perfectly accurate. These two deficiencies might bring in serious inconsistencies into the plan and mislead the planning to a wrong direction. They are the major factors that cause the unexpected bad performances of the hybrid system in the experiments.

6. POTENTIAL IMPROVEMENTS

With the experimental results, the hybrid system generally shows a better planning efficiency comparing to the original system. However, referring to the observed limitations of the current hybrid planning system, as well as the research objectives for the next stage, several potential improvements are expected in the future work.

1. A more intelligent sampling time. As observed, current random sampling time is not good enough and it is a major factor that causes bad performances of the case-based planning step. A better method is expected for improvements.
2. Online learning. This is the main objective in the next stage of research. Case-based planning is not only the approach which applies the past experience to solve current problems, but also the approach which learns from its past experience for future use. In the current stage, the learning is taken offline and all cases are predefined manually. Online learning, which enables runtime case generation in the phase of case retain, involves two major tasks, case feature generation and case solution generation. The first one is already accomplished in the current stage since the system is able to generate *PlanningSituation*, which is just the feature part of a case. The basic theoretical model for the second task is also established. More in-depth research and the implementation work of online learning are planned in the next stage of research.

3. A better method for case reuse. This method is expected to, firstly more intelligently handle the part of case solutions that cannot be instantiated, and secondly more effectively identify and prune away the useless part of a case solution.
4. Case base management. Case base management is to build up a compact case base which stores high quality cases with the minimal cost of memory. In the next stage, with an appropriate case evaluation mechanism combined with runtime case generation, a dynamic case base is expected to accept new and useful cases, and delete the useless ones. Eventually, a compact case can be established in the runtime. In the end of planning, all the maintained cases will be recorded into files for future use. (XML file is a good candidate to record cases.)
5. Parallel processing technology. For the purposes of runtime case generation and case base management, parallel processing technology can be applied. While the planning process keeps running in the front stage, these tasks are carried out by another process in the back stage. Some potential process conflicts can be avoided in this way.
6. Further experiments with various planning domains. One limitation of the current experiments is that, all the test problems are based on logistics domain. More experiments based on various planning domains are planned in the future work. Such experimental results can provide a more comprehensive

evaluation on this research and also inspire more ideas for further improvements.

7. CONCLUSIONS AND FUTURE WORK

The goal of this research is to integrate analogical reasoning as learning into general problem solving to handle planning problems more efficiently. For this purpose, an innovative hybrid planning approach that uses case-based planning to assist local-search planning is proposed. This hybridization is new in the literature and aims to take the advantages of both CBP and LSP through a dedicated design. The current experimental results validate the idea and support its potential to plan efficiently.

In the first step of my research, a case-based planner is developed to best fit the research purposes and the existing local-search planner. It applies structure-based feature representation along with vector-based index for case retrieve, models *output* (partial plan) as case solution, employs transformational adaptation technique for case reuse, creates a case ranking mechanism for case revise, and temporarily uses offline learning for case retain. The case-based planner is then integrated into the local-search planner in the form of a local-search heuristic (CBP heuristic). Inside the hybrid planner, the local-search planner triggers the case-based planner by calling the CBP heuristic to take a planning task, and it also assists the case-based planner to evaluate cases and accomplish its task of case revise. The hybrid planner was tested with classic logistics planning problems. The empirical results indicate that, comparing to the original local-search planning approach, the hybrid planning approach confirms a expected better performance in both planning speed and planning results.

With the analysis of the experimental results, the integrated case-based planner is likely to make such improvements from the following aspects.

- Firstly, the case-based planner remembers previous experiences as cases and uses them to guide the future planning process. The knowledge inside these cases seems very helpful to provide strong guidance in the planning by emphasizing some activities and temporarily ignoring the others. As a result, planning with the such guidance shows a better planning efficiency
- Secondly, with accurate identification of the target problems and working situations, and careful reuse of the previous solutions, the case-based planner usually provides high quality solutions that are expected by the target problem, e.g., organizing a sequence of wanted Move_Truck actions to transit a truck to a desired place. Solutions with better quality solve problems more effectively and also avoid the waste of further planning efforts. This may explain why the case-based planner performs helpful in pursuing better planning results.
- Lastly, the case-based planner itself works efficiently, especially in the phases of case retrieve and case reuse. With very careful designs, all these complicated tasks can be accomplished properly and considerably fast in most of the time. An efficient case-based planning component benefits the overall efficiency of the hybrid planner.

Besides the case-based planner, the local-search planner also contributes to these improvements from the following aspects.

- The local-search planner, which is the main planning process, provides an appropriate stage for the case-based planner. Firstly, the local-search planner progresses the main planning process using iterative repairs, and it is in charge of providing the resource required by the case-based planning step, e.g., some information about the current problem and situation. Secondly, the local-search planner recognizes the reliability of the heuristics and offers the case-based planner a considerable amount of opportunity to take planning tasks according to its remarkable performances. Especially, when the problem difficulty increases, the usage frequency of the CBP heuristic noticeably goes up.
- The local-search planner assists the case-based planner to perform the task of case revise. When sometimes the case-based planner suggests bad solutions, the local-search planner applies its strength to make effective repairs, e.g., removing, moving or modifying the inappropriately added actions.

The initial experimental results validate this innovative hybrid planning approach, and also promise a good potential to the idea behind that improving planning efficiency by integrating learning techniques into a general problem solver. Despite the achievements, this hybrid planning approach in the current stage has the following limitations:

- The case-based planner uses domain-dependent knowledge (cases) for planning. The learned knowledge for one domain is not applicable to solve problems from another domain. For example, the cases of transporting packages cannot be used to solve the problems of medical cares. Therefore, the case-based planner, or the hybrid planner, needs to prepare different case bases for different domains. The case bases are expected to be automatically generated in runtime, and this process will take extra computational efforts.
- In the current stage, using *output* (partial plan) to represent case solutions is only able to solve problems by adding new actions. This design is applied because it is more applicable to make online learning based on the current existing local-search planner. As a sequence, the application scope of the case-based planner is compromised. However, this might be a minor issue because the hybrid planner still has the local-search planner to provide the solutions in other forms, e.g., moving action, removing action, modifying action, etc.
- The hybrid planning approach has only been tested with logistics planning problems so far. More experiments based on various planning domains can be expected to provide more comprehensive evaluations.

In this research, a hybrid planner, which combines the local-search planner and the case-based planner, is proposed, designed, implemented and evaluated. The evaluation results indicate a good potential of this innovative hybrid planning approach to pursue a better planning efficiency in a dynamic and complex domain world. In the next stage, the main task of research is to employ online learning for the case-based planner to enable runtime case generation in the phase of case retain. Some other tasks planned in the future work include, the improvements on sampling time selection, better case reuse technique, the integration of case base management and the application of parallel processing technology (see Chapter 6 for more details). More experiments based various planning domains will certainly be taken for further evaluations.

BIBLIOGRAPHY

[1] A. Aamodt and E. Plaza. "Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Artificial Intelligence Communications,* v. 7, no. 1, 1994, pp. 39-52.

[2] R. Bergmann and W. Wilke, “PARIS: Flexible Plan Adaptation by Abstraction and Refinement,” In *Proceedings of the Workshop on "Adaptation in CBR", European Conference on Artificial Intelligence*, 1996.

[3] R. Bergmann and W. Wilke, “Building and Refining Abstract Planning Cases by Change of Representation Language”. *Journal of Artificial Intelligence Research,* 3, 1995, pp. 53-118.

[4] R. Bergmann, H. Munoz-Avila, M. M. Veloso, and E. Melis, “Case-based reasoning Applied to Planning”. In M. Lenz, B. Bartsch-Sporl, and S. Wess, editors, *Case-based Reasoning Technology from Foundations to Applications*, volume 1400 of Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin, Germany, 1998, pp. 169-203.

[5] R. Bergmann, J. Kolodner and E. Plaza, “Representation in Case-based Reasoning,” *The Knowledge Engineering Review*, Vol. 00:0, 2005, pp. 1-4.

[6] A. Blum, and M. Furst, “Fast Planning through Planning Graph Analysis,” *Artificial Intelligence,* 90, 1997, pp. 281–300.

[7] D. Chapman, “Planning for conjunctive goals,” *Artificial Intelligence*, 32, 1987, pp. 333-377.

[8] K. Erol, D. S. Nau and V. S. Subrahmanian, Complexity, Decidability and Undecidability Results for Domain-Independent Planning. Technical Report CS-TR-2797, University of Maryland, Institute for Advanced Computer Studies, Maryland, USA, 1991.

[9] R. E. Fikes and N. J. Nilsson, “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving,” *Artificial Intelligence*, 2(3-4), 1971, pp. 189-208.

[10] H. Geffner, “Perspectives on Artificial Intelligence Planning, ” in *Proceedings Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, 2002, AAAI/MIT Press, pp. 1013-1023.

[11] N. Gupta and D. S. Nau, “On the complexity of blocks-world planning,” *Artificial Intelligence* 56(2-3), 1992, pp. 223–254.

[12] A. K. Goel, K. S. Ali and E. Stroulia, *Some Experimental Results in Multistrategy Navigation Planning*, Technical Report GIT-CC-95-51, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, 1995.

[13] K. J. Hammond, “Case-based Planning: a Framework for Planning from Experience,” *Cognitive Science,* 14(3), July 1990, pp. 385-443.

[14] S. Hanks and D. Weld. “A Domain-Independent Algorithm for Plan Adaptation,” *Journal of Artificial Intelligence Research*, 2, 1995, pp. 319-360.

[15] J. Hendler, A. Tate, and M. Drummond, “AI planning: Systems and Techniques,” *AI magazine*, vol. 11, Summer 1990, pp. 61-77.

[16] L. G, Ihrig and S. Kambhampati, *Plan-space vs State-space Planning in Reuse and Replay*, Technical Report ASU CSU TR 94-006, Department of Computer Science and Engineering, Arisona State University, Tempe, AZ, December 1996.

[17] L. H. Ihrig and S. Kambhampati, “Storing and Indexing Plan Derivations through Explanation-based Analysis of Retrieval Failures,” *Journal of Artificial Intelligence Research*, 7, 1997, pp. 161-198.

[18] P. Liberatore, “On the Complexity of Case-based Planning,” *Journal of Experimental & Theoretical Artificial Intelligence*, 1362-3079, 17(3), 2005, pp. 283 – 295.

[19] I. Jurisica, Representation and Management Issues for Case-Based Reasoning Systems, Department of Computer Science, University of Toronto, Toronto, Ontario M5S1A4, Canada (Sept 1993).

[20] S. Kambhampati and J. Hendler. “A Validation-Structure-based Theory of Plan Modification and Reuse,” *Artificial Intelligence*, 55, 1992, pp. 193-258.

[21] H. Kautz and B. Selman, “Planning as Satisfiability,” In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI_92)*, 1992, pp. 359-363.

[22] H. Kautz and B. Selman, “Pushing the Envelop: Planning, Propositional Logic, and Stochastic Search,” In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996, pp. 1194-1201.

[23] B. P. Kettler, J. Hendler, W. A. Andersen, and M. P. Evett, “Massively Parallel Support for Case-based Planning,” *IEEE Expert*, February 1994, pp. 8-14.

[24] Y. A. Kochetov, “Computational Bounds for Local Search in Combinatorial Optimization,” *Computational Mathematics and Mathematical Physics*, 48(5), 2008, pp. 747-763.

[25] M. Manago, R. Bergmann, S. Wess, and R. Traph¨oner, *CASUEL: A Common Case Representation Language*, ESPRIT Project INRECA, No. 6322, Deliverable D1, University of Kaiserslautern, Kaiserslautern Germany, 1994.

[26] A. Nareyek. “Local-search heuristics for generative planning,” In *Fifteenth Workshop on AI in Planning, Scheduling, Configuration and Design*, 2001, pp. 56–70.

[27] A. Nareyek, *Crackpot*, Website, 2008, http://sourceforge.net/projects/crackpot/.

[28] E. Plaza, F. Esteva, P. Garcia, L. Godo, and R. Lopez de Mantaras. “A Logical Approach to Case-based Reasoning Using Fuzzy Similarity Relations,” *Information Science*, 1997.

[29] B. W. Porter, R. Bareiss, and R. C. Holte, “Concept Learning and Heuristic Classification in Weak-Theory Domains,” *Artificial Intelligence*, 45, March 1990, pp. 229-263.

[30] F. Ricci and P. Avesani. “Learning a Local Similarity Metric for Case-based Reasoning,” In *Proceedings of the 1st International Conference on Case-based Reasoning Research and Development*, volume 1010 of Lecture Notes in Artificial Intelligence, Berlin, October, 23-26 1995. Springer Verlag.

[31] K. E. Sanders, B. P. Kettler, and J. Hendler, “The Case for Graph-Structured Representation*,” In Proceedings of the First International Conference on Case-based Reasoning*. 1995. Springer Verlag.

[32] L. Spalazzi, “A Survey on Case-based Planning,” *Artificial Intelligence Review*, 16(1), 2001, pp. 3-36.

[33] C. Tsatsoulis and R. L. Kashyap, “Case-based Reasoning and Learning in Manufacturing with the TOLTEC Planner,” *IEEE Transations on Systems, Man, and Cybernetics*, 23(4), July 1993, pp. 1010-1023.

[34] M. M. Veloso. “Flexible Strategy Learning: Analogical Reply of Problem Solving Episodes,” In *Proceedings of AAAI-94*, AAA Press, 1994, pp. 595-600.

[35] M. Veloso, J. Carbonell, A. Prerez, D. Borrajo, E. Fink and J. Blythe. “Integrating Planning and Learning: The PRODIGY Architecture,” *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1), 1995.

[36] M. M. Veloso, “Prodigy/Analogy: Analogical Reasoning in General Problem Solving,” In: Wess, S., Richter, M., Althoff, K.-D. (eds.) *Topics in Case-Based Reasoning*, LNCS, vol. 837, Springer, Heidelberg, 1994, pp. 33–52.

[37] M. M. Veloso, H. Munoz-Avila, and R. Bergmann, “Case-based Planning: Selected Methods and Systems,” *AI Communications*, 9(3), September 1996, pp. 128-137.

[38] J. Zhang and H. Zhang, “Combining Local Search and Backtracking Techniques for Constraint Satisfaction”. In *Proceedings of AAAI-96*, AAAI Press/MIT Press, 1996, pp. 369-374.