Integrando RosPlan com FastDownward

Marcelo Paravisi
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - IFRS Campus Osório

Abstract

This document presents the final report of Automatic Planning Course. The main purpose of this paper is to replace the planner RosPlan by Fast Downward planner and perform a study case in a virtual ambient. To validate the implementation was conducted an experiment, the generated data is analysed in a quantitative and qualitative way.

Introdução

Este documento descreve como foi realizada a atividade final da disciplina de Planejamento Automático do Programa de Pós-Graduação em Ciência da Computação, ofertada pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) no segundo semestre de 2016. O objetivo principal desse projeto é a substituição do sistema de planejamento do RosPlan (ea15) pelo planejador Fast Downward (Hel06).

O RosPlan é uma ferramenta capaz de planejar tarefas a serem realizadas num sistema ROS, sendo que o RosPlan contempla tanto o planejamento quanto o envio para execução dos planos gerados. Para isso, o RosPlan é constituído por um sistema de planejamento de 4 etapas: *Problem Generation*, *Planner*, *Plan Parser* e *Plan Dispatch*. Esse sistema é responsável por transformar as informações contidas no *Knowledge Base* em ações a serem executadas pela plataforma ROS. Contudo, forma-se um ciclo completo, pois essa plataforma obtem informações do estado atual do ambiente e alimenta o *Knowledge Base*. Na Figura 1, é apresentada o diagrama de funcionamento do ROSPlan.

Já o sistema de planejamento Fast Downward é um planejador progressivo, em que a busca ocorre em *foward direction*. Portanto, o planejador parte do estado inicial e vai considerando as ações para atingir um objetivo. Ao longo deste processo, utiliza um custo, podendo ser unitário ou definido pelo usuário por meio da extensão pddl *:action-cost*. Além disso, ele faz uma decomposição hierarquica para calcular a sua função heurística, sendo ela conhecida como *causal graph heuristic*. Essa função é diferente das heurísticas baseadas em HSP.

O algoritmo Fast Downward é realizado em três fases: tradução, compilação de conhecimento e busca. Na tradução, transforma-se a entrada descrita em PDDL 2.2 para uma tarefa de planejamento multi-valorada. Já na compilação de conhecimento, gera 4 tipos de informações para serem utilizadas na fase de busca: domain transition graphs, causal graph, successor generator, axiom evaluator. Abaixo está descrito o significado de cada tipo de informação gerado na fase de compilação de conhecimento:

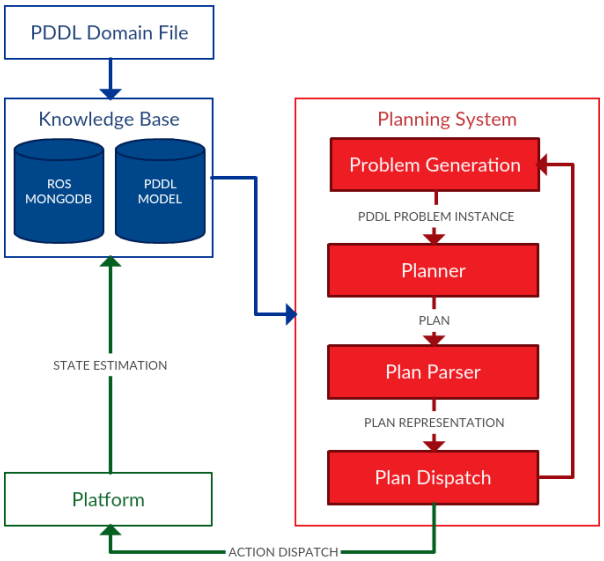

Figura 1: Estrutura do framework ROSPlan.
Fonte: `http://kcl-planning.github.io/ROSPlan/documentation/`

- *Domain transition graphs*: define as condições para as variáveis de estado mudam de estado.
- *Causal graph*: grafo com as depedências hierarquicas entre variáveis de estado.
- *Sucessor generator*: estrutura de dados eficiente para determinar o conjunto de operadores a serem aplicados a um estado específico.
- *Axiom evaluator*: estrutura de dados eficiente para calcular os valores provinentes das variáveis.

Na fase de busca, são implementados três algoritmos de busca para realizar o planejamento:

- *Greedy best-first search*: utiliza uma heurística de grafo causal.
- *Multi-heuristic best-first search*: variação do algoritmo anterior, sendo que combina heurísticas do *causal graph* e *FF heuristics*.
- *Focused iterative-broadening*: Não utiliza uma função de avaliação heurística, mas verifica no grafo causal quais operadores não ajudarão a atingir o objetivo de uma tarefa.

Para melhor compreensão, o presente relatório encontra-se organizado em seções, no qual estão dispostos na sequência: abordagem técnica abrangendo a solução utilizada; Experimentos apresentando os resultados obtidos; Conclusão com uma visão dos resultados que poderão ser obtidos.

Abordagem técnica

Para realizar a substituição do planejador ROSplan, de acordo com a documentação do mesmo, é necessário a realização de até 3 etapas:

- Alterar o arquivo de inicialização do RosPlan, informando o comando necessário para executar o Fast Downward com o domínio e problema propostos;
- Converter as saídas do Fast Downward para a estruturas do RosPlan (Parser);
- Conversão da saída do problema gerado pelo ROSPlan para a sintaxe aceita pelo FastDownward.

O projeto foi realizado em diversas etapas, partindo-se da instalação do planejador Fast Downward e do RosPlan. Nessa etapa do projeto, encontraram-se incompatibilidades das bibliotecas do ROSPlan com o Ubuntu 16.04 e com o ROS Kinetic. Desta forma, a solução encontrada foi a realização de *downgrade* do Sistema Operacional para a versão 14.05 e do ROS para a versão Indigo.

Após realizada a instalação completa do ROSPlan no Ubuntu 14.05 e da realização de experimento de teste disponibilizado pelo ROSPlan, foi iniciada a substituição do planejador do RosPlan pelo Fast Downward. A subtituição foi realizada por meio da alteração do arquivo de inicialização do RosPlan (interfaced_planning_system.launch), informando o comando necessário para executar o Fast Downward com o domínio e problema propostos.

O Fast Downward tem como saída na tela (stdout) um log de informações sobre como foi realizado planejamento e o plano gerado é armazenado num arquivo específico (plan.pddl). Já o ROSPlan espera que a saída na tela contenha o plano gerado. Assim, o ROSPlan sobreescreve o arquivo plan.pddl com a saída na tela do planejador. O que fazia com que o plano gerado pelo Fast Downward fosse apagado. Desta forma, foi necessário alterar o código fonte do ROSPlan para que levasse em consideração apenas o arquivo gerado (plan.pddl) e não mais a saída do planejador. Abaixo pode ser observada o comando com o planejador padrão do ROSPlan e o comando para o FastDownward.

Após realizada essa adequação, foi necessário converter as saídas do Fast Downward para as estruturas do RosPlan. Para isso, foi necessário criar um parser do plano gerado pelo Fast Downward. Ao construir o parser, observou-se que o ROSPlan permite a definição do tempo de duração de cada ação e o instante em que a ação deverá ser dispachada para ser publicada nos tópicos do ROS. Contudo, isso pode ser realizado por meio da extensão pddl :durative-actions, porém ela não é compatível com o planejador Fast Downward. Como solução parcial, foram removidas os marcadores de tempo de realização *at start* e *at end* dos efeitos e condições das ações do domínio do problema.

Além disso, o ROSPLan utiliza a extensão pddl :fluents para a definição de valores de distância. Contudo, essa extensão não é compatível com o FastDownward. Desta forma, a solução encontrada para isso foi a utilização de valores inteiros, ou seja as distâncias passaram a ser definidas em centimetros e não mais em metros. Para tal, foi necessário alterar o gerador de problemas do ROSPlan (classe PDDLProblemGenerator).

Por fim, foi realizada uma experimentação da solução obtida das etapas anteriores. Ela é apresentada na próxima seção.

Experimento

Para realização deste projeto, um estudo de caso de conduzido em que os dados foram gerados por meio de experimentação. Essa experimentação foi realizada utilizando-se um ambiente virtual de simulação, que através dos dados levantados foram realizadas análises do tipo mista, ou seja, qualitativamente e quantitativamente.

Essa etapa de experimentação foi constituída pela definição de um problema a ser solucionado e pela realização de testes. Foi utilizado o problema disponibilizado pelo ROSPLan, em que uma série de pontos num ambiente de simulação são sorteados e o planejador deve encontrar um plano de forma que um *turtlebot* percorra todos os pontos.

Para realizar a execução no ROSPLan (consequentemente do experimento), os comandos abaixo devem ser executados.

1. roscore
2. sudo service mongodb stop
3. source /opt/ros/indigo/setup.bash
4. source devel/setup.bash
5. roslaunch rosplan_demos turtlebot2.launch
6. rqt - -standalone rosplan_rqt.dispatcher.ROSPlanDispatcher
7. sh src/rosplan/rosplan_demos/scripts/turtlebot_explore.bash

Os comandos 1 e 2 devem ser executados uma única vez na inicialização do sistema operacional. Já os comandos 3 e 4 devem ser executados em cada terminal uma única vez antes da execução dos comandos 5 a 7. Portanto, cada um dos comandos de 5 a 7 devem ser executados em terminais diferentes. O comando 5 inicia o ambiente de simulação, sendo que serão apresentados as janelas do Rviz e do gazebo. Já

o comando 6 abrirá uma interface de visualização do ROSPlan, na qual poderá acompanhar os estados da execução do plano gerado. Já o comando 7 publicará nos tópicos do ROS informações necessárias para construção de conhecimento no MongoDB e inicialização do planejamento.

O experimento foi conduzido com 4 combinações de algoritmos de busca e heurística: A* com heurística blind, A* com heurística ipdb, greedy com heurística FF e Context Heuristic, greedy com heurística FF. O objetivo era percorrer 5 *waypoints* sorteados dinamicamente num ambiente plano contendo obstáculos. O ROSPlan faz o cruzamento com a informação do ambiente de forma a saber quais *waypoints* estão conectados. Na Figura 2, pode-se observar turtlebot (robô) no centro no ambiente composto por 5 obstáculos.

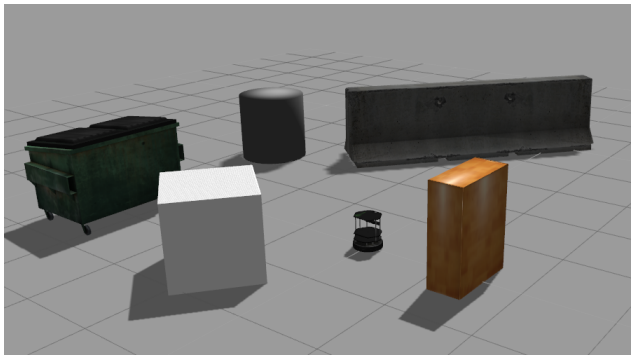

Figura 2: Ambiente de simulação utilizado no experimento. Fonte: próprio autor.

Todos os algoritmos geraram o mesmo plano e tiveram um pico de memória de 3648 KB. O plano gerado pode ser visualizado abaixo, sendo que a ação goto_waypoint move o robô conhecido como kenny de um waypoint de origem para um waypoint destino.

```
goto_waypoint kenny wp0 wp0 (1)
goto_waypoint kenny wp0 wp1 (1)
goto_waypoint kenny wp1 wp2 (1)
goto_waypoint kenny wp2 wp3 (1)
goto_waypoint kenny wp3 wp4 (1)
```

Além dessas informações, foram coletados os tempos para criação do plano, sendo que essa informação é apresentada na Tabela 1. O único método de busca que consumiu mais tempo foi o A* com heurística blind. As outras heurísticas tiveram um tempo similar para planejamento.

Algoritmo de Busca	Heurística	Tempo (s)
A*	blind	0,0014
A*	IPDB	0,0006
Greedy	FF and Context Heurístic	0,0006
Greedy	ff	0,0006

Tabela 1: Tempo para gerar o plano para cada algoritmo de busca no Fast Downward.

Outro experimento realizado foi a de criação de maior número de *waypoints* no ambiente da Figura 2. Além disso, foi realizado um relaxamento nas regras de criação de *waypoints* existentes na classe RPRoadmapServer. Desta forma, foi tomada como hipótese que todos os waypoints estavam interligados, ou seja, pode-se mover de um waypoint para qualquer outro waypoint. Após realizada a simulação, observou-se que os planos gerados levavam o robô até um obstáculo. Nesses casos, o plano era tido com falha, necessitando um novo planejamento.

Como último experimento, foram colocados pontos no ambiente em formato de grade conforme a Figura 3. Nesse experimento, foi configurado de forma que o *turtlebot* percorre-se todos os pontos no ambiente. Para tal, foi necessário adicionar uma métrica ao problema de planejamento proposto. Essa métrica objetivava minimizar a distância percorrida para percorrer todos os nós. Com esse experimento, foi possível realizar uma simplificação de Coverage Path Planning por meio do planejador Fast Downward e pelo ROSPlan.

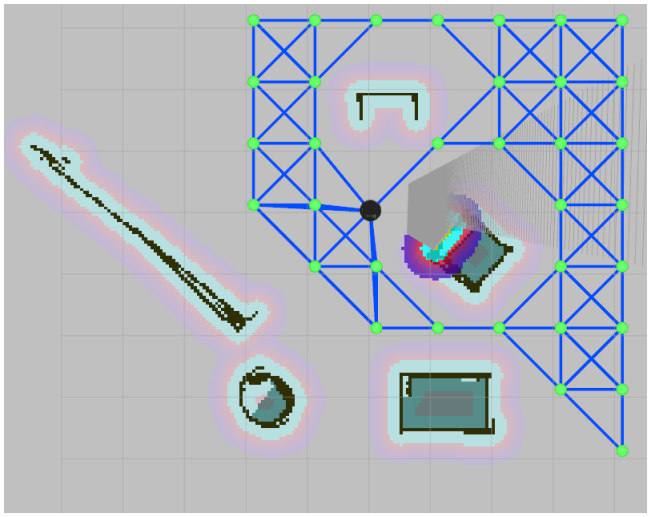

Figura 3: Ambiente do último experimento. Circulos verdes são os waypoints em que o *turtlebot* deverá visitar.
Fonte: próprio autor.

Conclusão

Por meio deste projeto pode-se consolidar os conhecimentos obtidos na disciplina de Planejamento Automático, aproximando ainda mais com o tema de pesquisa e ferramentas a serem desenvolvidas na tese de doutorado do autor dessa proposta. Destacando-se o aprofundamento dos conhecimentos em ROS, planejamento automático e ROSPlan.

Além disso, a realização dessa atividade permitiu compreender melhor como ocorre a integração de um sistema planejamento com dispositivos disparadores e executores das ações providas pelo *Fast Downward*. Conseguindo assim, observar a diferença entre um sistema planejador simples e um sistema planejador dinâmico, no qual os planos são gerados utilizando-se também de informações do estado atual.

Observa-se assim que os sistemas dinâmicos constituem-se em ciclo fechado, em que as informações são colhidas do ambiente e da execução das ações planejadas, de forma a atualizar o plano gerado (Gha04).

Como trabalho futuro, espera-se avaliar o funcionamento do ROSPlan frente a alteração do conhecimento geometrico. Um exemplo de estudo de caso a ser desenvolvido é aquele em que o ambiente tem uma grande extensão e que a medida que o robô move-se, a informação do ambiente é alterada. Provocando assim, uma alteração no conhecimento e no estado inicial do problema fornecido ao planejador.

Uma outra atividade a ser realizada no futuro compreende a adição de outras métricas além da distância para realizar o Coverage Path Planning do último experimento. Um exemplo de experimento a ser realizado pode ser por meio da adição de inclinação ao ambiente de simulação.

References

[ea15] Cashmore et al. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 333–341, 2015.

[Gha04] Dana; Paolo Traverso Ghallab, Malik; Nau. *Automated Planning: theory and practice*, volume 1 of *1*. Elsevier, San Francisco, 1ª edition, 2004.

[Hel06] Malte Helmert. The fast downward planning system. volume 26, pages 191–246, USA, July 2006. AI Access Foundation.