Artículo científico de planificadores del estado del arte.

Balsells Orellana, Jorge A.

4 de enero de 2021

Resumen

Este documento contiene los resultados de análisis de algunos planificadores existentes en IPC del año 2018. IPC es una competencia internacional anual del estado de arte de planificadores, en la cual se motiva a desarrolladores a publicar sobre sus investigaciones en esta rama de la inteligencia artificial. Se han instalado y probado los primeros lugares en cada rama, y se han seleccionado 3 al azar para analizarlos de la misma manera y comparar sus resultados plasmados en conclusiones.

1. Introducción

La planificación en Inteligencia Artificial es una forma automática de programación a través de una secuencia de acciones que parten desde un estado inicial, hacia un estado final. En este caso se aplica al *estado del arte*, lo que significa que es una investigación técnica y científica en el campo de inteligencia artificial. Los planificadores no solo dan la solución a un problema, sino los pasos a seguir dentro del problema, por este motivo son tan importantes dado que se da a conocer el proceso con pasos ordenados y no solo el resultado obtenido.

Otro aspecto muy importante de la planificación automática son las técnicas existentes para planificación y la fuerte semejanza entre los procesos de planificación y los procesos de programación comparando el concepto de un plan y el desarrollo de un algoritmo.

2. Estado del arte

Es una categoría deductiva que nace con la pretensión de generar un balance en la investigación de algunas regiones, y tiene como fin una estrategia metodológica para análisis crítico en cualquier área de trabajo o estudio.Requiere un análisis hermenéutico del objeto de estudio sin ningún tipo de problema ante la crítica, para poder superar o igualar estudios existentes por procesos diferentes a lo ya existente. (Guevara Patiño, 2016)

En otras palabras, se puede decir que es una investigacion documental que sirve para compartir el conocimiento que se ha generado sobre un estudio específico. Esto hace posible una mejor comprensión crítica sobre un tema con fin de generar nuevos conocimientos y nuevas comprensiones. Permite desarrollar nuevas teorías a partir de análisis crítico, revisión e interpretación de los datos y documentos existentes. En este caso, la inteligencia artificial se ha centrado en razonamiento sobre la planificación de enfrentar problemas y resolverlos, y el estado del arte aplica en los algoritmos utilizados para ello. (Guzmán Luna, 2003)

3. Planificadores evaluados

En la selección de los planificadores automáticos de este documento, están basados en determinar cuáles tienen experimentos en su repositorio. Los planificadores seleccionados han sido de la competicion internacional de planificacion de 2018. Esta competición evalúa el estado del arte de planificadores cada año, siendo asi un medio que promueve el desarrollo de estos.

3.1. Fast Downward Stone Soup

Este planificador requiere un set de algoritmos de planificación que usa 144 algoritmos con configuraciones *fast downward*. De igual manera necesita instancias de entrenamiento, específicamente 2115 instancias.

```
build-portfolio(algorithms,results,
    granularity, timeout):
portfolio := {A 7    0 | A ...
        algorithms}
repeat btimeout/granularityc ...
    times:
candidates := ...
    successors(portfolio, ...
    granularity)
portfolio := arg ...
     m a x C candidates  ...
    score(C,results)
portfolio := ...
    reduce(portfolio,results)
return portfolio
```

La evaluación de resultados del algoritmo tiene 2 parámetros, *Granularity* y *Timeout* medidos en segundos. Ruta Satisfactoria(*Satisficing Track*) (R. G. Seipp Jendrik, 2018)

Los resultados obtenidos de este algoritmo fueron todos los mensajes generados durante la ejecución, en un documento de texto de **1605673** líneas descritas, donde las ultimas 5 líneas son las siguientes.

```
1605669 3
1605670 11 1
1605671 22 1
1605672 39 1
1605673 end_CG
```

y luego de aproximadamente 30 minutos, en terminal ha quedado el siguiente mensaje:

```
remaining time: -0.72
config 16: relative time 30, ...
    remaining 210
Error: Unexpected exit codes: ...
    [-11]
Command '['run-portfolio', ...
    '/planner/driver/ ...
    portfolios/seq_sat_fdss_2018 ...
    .py']' returned non-zero ...
    exit status -11
```

3.2. LAPKT BFWS Preference

Se utilizan algoritmos basados en *búsqueda ancha* que obtienen la noción de un estado, suponiendo que el estado es el conjunto de proposiciones. La búsqueda de mejor ancho primero proporciona una alternativa a IW(k) serializados completos, lo que implica que se pierde la naturaleza polinomial - temporal de IW(k). Ruta Satisfactoria(*Satisficing Track*) (Francés Guillem, 2018)

Los resultados obtenidos son los siguientes:

```
(take_grain worker2 worker1 ...
   worker2 round1 grain)
(plow_field worker1 noworker ...
   worker2 round1)
(ag__finish_round_backhome ...
   round1 worker2)
(ag__finish_round_renew ...
   round1 noworker)
(ag__advance_round_normal ...
   round1 round2 act_sheep)
(take_grain worker2 worker1 ...
   worker2 round2 carrot)
(take_food worker1 noworker ...
   worker2 round2 num2 num3)
(ag__finish_round_backhome ...
   round2 worker2)
(ag__finish_round_renew ...
   round2 noworker)
(ag__advance_round_normal ...
   round2 round3 act_sow)
(take_food worker2 worker1 ...
   worker2 round3 num3 num4)
(sow worker1 noworker worker2 ...
   round3 grain)
(ag__finish_round_backhome ...
   round3 worker2)
(ag__finish_round_renew ...
   round3 noworker)
(ag__advance_round_normal ...
   round3 round4 act_fences)
(take_food worker2 worker1 ...
   worker2 round4 num4 num5)
(collect_resource worker1 ...
   noworker worker2 round4 ...
   act_clay clay)
(ag__finish_round_backhome ...
   round4 worker2)
(ag__finish_round_renew ...
   round4 noworker)
(ag__harvest_collect_end ...
   round4 stage1)
(ag__harvest_feed round4 ...
   stage1 worker2 num5 num5 num0)
(ag__harvest_breed_end round4 ...
   stage1)
(ag__finish_stage stage1 ...
   stage2 round4 round5 act_boar)
(take_food worker2 worker1 ...
   worker2 round5 num0 num1)
(collect_resource worker1 ...
   noworker worker2 round5 ...
   act_stone stone)
(ag__finish_round_backhome ...
   round5 worker2)
(ag__finish_round_renew ...
   round5 noworker)
(ag__advance_round_normal ...
   round5 round6 act_improve)
(improve_home worker2 worker1 ...
   worker2 round6 fireplace)
(collect_cook_animal boar ...
   act_sheep worker1 noworker ...
   worker2 round6 num1 num3)
(ag__finish_round_backhome ...
   round6 worker2)
(ag__finish_round_renew ...
   round6 noworker)
(ag__advance_round_normal ...
   round6 round7 act_cattle)
(collect_cook_animal boar ...
   act_sheep worker2 worker1 ...
   worker2 round7 num3 num5)
(collect_cook_animal boar ...
   act_cattle worker1 ...
   noworker worker2 round7 ...
   num5 num7)
(ag__finish_round_backhome ...
   round7 worker2)
(ag__finish_round_renew ...
   round7 noworker)
(ag__harvest_collect_end ...
   round7 stage2)
(ag__harvest_feed round7 ...
   stage2 worker2 num7 num5 num2)
(ag__harvest_breed_end round7 ...
   stage2)
```

```
(ag__finish_stage stage2 ...
    stage3 round7 round8 ...
    act_carrot)
(collect_cook_animal boar ...
    act_sheep worker2 worker1 ...
    worker2 round8 num2 num4)
(sow worker1 noworker worker2 ...
    round8 carrot)
(ag__finish_round_backhome ...
    round8 worker2)
(ag__finish_round_renew ...
    round8 noworker)
(ag__advance_round_normal ...
    round8 round9 void)
(take_food worker2 worker1 ...
    worker2 round9 num4 num5)
(collect_cook_animal boar ...
    act_sheep worker1 noworker ...
    worker2 round9 num5 num7)
(ag__finish_round_backhome ...
    round9 worker2)
(ag__finish_round_renew ...
    round9 noworker)
(ag__harvest_collect_end ...
    round9 stage3)
(ag__harvest_feed round9 ...
    stage3 worker2 num7 num5 num2)
(ag__harvest_breed_end round9 ...
    stage3)
```

3.3. Complementary 2

Este planificador es una implementación de las heurísticas (CPC). Es una descripción general de alto nivel del CPC de búsqueda realizada en un espacio de colección de patrones. Recibe una tarea de planificación con una base heurística, límites de tiempo y memoria. Este devuelve un conjunto de colecciones de patrones en base a una función heurística canónica. Ruta óptima(*Optimal Track*) (Franco Santiago, 2018)

Los resultados obtenidos son los siguientes:

```
(collect_resource worker2 ...
    worker1 worker2 round1 ...
    act_reed reed)
(collect_resource worker1 ...
    noworker worker2 round1 ...
    act_wood wood)
(ag__finish_round_backhome ...
    round1 worker2)
(ag__finish_round_renew ...
    round1 noworker)
(ag__advance_round_normal ...
    round1 round2 act_sheep)
(build_room worker2 worker1 ...
    worker2 worker3 round2 room3)
(family_growth worker1 ...
    noworker worker2 worker3 ...
    round2 clay room3)
(ag__finish_round_backhome_withchild ...
    round2 worker2 worker3)
(ag__finish_round_renew ...
    round2 noworker)
(ag__advance_round_normal ...
    round2 round3 act_sow)
(collect_resource worker3 ...
    worker2 worker3 round3 ...
    act_clay clay)
(collect_resource worker2 ...
    worker1 worker3 round3 ...
    act_reed reed)
(plow_field worker1 noworker ...
    worker3 round3)
(ag__finish_round_backhome ...
    round3 worker3)
(ag__finish_round_renew ...
    round3 noworker)
(ag__advance_round_normal ...
    round3 round4 act_fences)
(take_food worker3 worker2 ...
    worker3 round4 num2 num3)
(take_grain worker2 worker1 ...
    worker3 round4 carrot)
(sow worker1 noworker worker3 ...
    round4 carrot)
(ag__finish_round_backhome ...
    round4 worker3)
(ag__finish_round_renew ...
    round4 noworker)
(ag__harvest_collecting_veg ...
    round4 stage1 carrot num3 ...
    num5 num6)
(ag__harvest_collect_end ...
    round4 stage1)
(ag__harvest_feed round4 ...
    stage1 worker3 num6 num6 num0)
(ag__harvest_breed_end round4 ...
    stage1)
(ag__finish_stage stage1 ...
    stage2 round4 round5 act_boar)
(collect_resource worker3 ...
    worker2 worker3 round5 ...
    act_wood wood)
(build_room worker2 worker1 ...
    worker3 worker4 round5 room4)
(family_growth worker1 ...
    noworker worker3 worker4 ...
    round5 clay room4)
(ag__finish_round_backhome_withchild ...
    round5 worker3 worker4)
(ag__finish_round_renew ...
    round5 noworker)
(ag__advance_round_normal ...
    round5 round6 act_improve)
(collect_resource worker4 ...
    worker3 worker4 round6 ...
    act_stone stone)
(improve_home worker3 worker2 ...
    worker4 round6 fireplace)
(collect_cook_animal boar ...
    act_sheep worker2 worker1 ...
    worker4 round6 num0 num2)
(take_food worker1 noworker ...
    worker4 round6 num2 num3)
(ag__finish_round_backhome ...
    round6 worker4)
(ag__finish_round_renew ...
    round6 noworker)
(ag__advance_round_normal ...
    round6 round7 act_cattle)
(build_fences boar worker4 ...
    worker3 worker4 round7)
(collect_cook_animal boar ...
    act_boar worker3 worker2 ...
    worker4 round7 num3 num5)
(collect_cook_animal boar ...
    act_sheep worker2 worker1 ...
    worker4 round7 num5 num7)
(take_food worker1 noworker ...
    worker4 round7 num7 num8)
(ag__finish_round_backhome ...
    round7 worker4)
(ag__finish_round_renew ...
    round7 noworker)
(ag__harvest_collect_end ...
    round7 stage2)
(ag__harvest_feed round7 ...
    stage2 worker4 num8 num8 num0)
(ag__harvest_breed_end round7 ...
    stage2)
(ag__finish_stage stage2 ...
    stage3 round7 round8 ...
    act_carrot)
(build_fences boar worker4 ...
    worker3 worker4 round8)
(collect_cook_animal boar ...
    act_cattle worker3 worker2 ...
    worker4 round8 num0 num2)
(collect_cook_animal boar ...
    act_sheep worker2 worker1 ...
    worker4 round8 num2 num4)
(take_food worker1 noworker ...
    worker4 round8 num4 num5)
(ag__finish_round_backhome ...
    round8 worker4)
(ag__finish_round_renew ...
    round8 noworker)
(ag__advance_round_normal ...
    round8 round9 void)
(build_fences boar worker4 ...
    worker3 worker4 round9)
(collect_cook_animal boar ...
    act_sheep worker3 worker2 ...
    worker4 round9 num5 num7)
(collect_resource worker2 ...
    worker1 worker4 round9 ...
    act_clay clay)
(take_food worker1 noworker ...
    worker4 round9 num7 num8)
(ag__finish_round_backhome ...
    round9 worker4)
(ag__finish_round_renew ...
    round9 noworker)
(ag__harvest_collect_end ...
    round9 stage3)
```

3.4. Symple

Planificación simbólica basada en EVMDDs Edge-Valued Multi-Valued Decision Diagrams. Este fué el primer planificador que ejecuté, sin embargo no fué satisfactoria la ejecución, ya que la demora generada era superior al límite que tienen en IPC para poder ejecutar un algoritmo, y no fué funcional en mi sistema el paquete, ya que generaba errores de sintaxis. Sin embargo, se decidió dejar este problema en el documento ya que si fue instalado, pero se optó por considerar un planificador más. Ruta óptima(*Optimal Track*) (Speck David, 2018).

3.5. Scorpion

Planificador abstracto de optimizacion secuencial de rutas implementado en sistemas rapidos de planificacion descendente con Fast Downward. Fast downward es un planificador clasico de dominio independiente, en el cual el mismo autor de Scorpion tiene contribuciones. Ruta óptima(*Optimal Track*)(J. Seipp, 2018).

Este planificador genera errores en la ejecución, por lo cual la única salida retornada son los mensajes generados a través de la ejecución del script. Los cuales en total son 276117. Siendo los últimos 5 los siguientes:

```
276113  3
276114  11 1
276115  21 1
276116  39 1
276117  end_CG
```

4. Instalación de planificadores

Los planificadores se utilizan de la misma manera, motivo por el cual solamente se describen las instrucciones de instalación generalizadas para los planificadores.

Para utilizar los planificadores, necesitamos Singularity, que es una tecnología para crear contenedores virtuales con contenido y hacer mas eficiente la virtualización. Singularity utiliza un propio formato de contenedores, pero permite la importación de contenedores de otras plataformas como Docker. Para instalar Singularity, necesitamos instalar Go, que es un lenguaje de programación concurrente y compilado que busca ser una opción con el rendimiento de C y la dinámica de Python.

Singularity se ha instalado en Ubuntu 20.04, teniendo en cuenta las dependencias previas a la instalación.

```
sudo apt-get update && sudo ...
    apt-get install -y \
     build-essential \
     uuid-dev \
     libgpgme-dev \
     squashfs-tools \
     libseccomp-dev \
     wget \
     pkg-config \
     git \
     cryptsetup-bin
```

Luego de instalar las dependencias, se procede a instalar GO y a crear su variable de entorno respectiva.

```
export VERSION=1.13.5 ...
    OS=linux ARCH=amd64 && \
     wget ...
```

```
        https://dl.google.com/ ...
        go/go$VERSION ...
        .$OS-$ARCH.tar.gz && \
    sudo tar -C /usr/local ...
        -xzvf go$VERSION ...
        .$OS-$ARCH.tar.gz && \
    rm ...
        go$VERSION.$OS-$ARCH.tar.gz

echo 'export ...
    GOPATH=${HOME}/go' >> ¬...
    /.bashrc && \
    echo 'export ...
        PATH=/usr/local/go/bin ...
        :${PATH} ...
        :${GOPATH}/bin' >> ¬...
        /.bashrc && \
    source ¬/.bashrc
```

Teniendo lo anterior listo, se procede a revisar la versión de descarga y a descargar singularity como usuario root, y por último realizamos los cambios necesarios para poder trabajar con singularity con un usuario sin privilegios de administrador.

```
git clone ...
    https://github.com/sylabs ...
    /singularity.git && \
    cd singularity && \
    git checkout v3.5.2

./mconfig && \
    make -C ./builddir && \
    sudo make -C ./builddir ...
        install

./mconfig ...
    --prefix=/opt/singularity

./mconfig --without-setuid ...
    --prefix=/home/jbalsells ...
    /singularity && \
    make -C ./builddir && \
    make -C ./builddir install
```

En el repositorio de IPC 2018 *https://ipc2018-classical.bitbucket.io/* están las instrucciones para poder correr los scripts en la competencia. Estos mismos nos sirven para correr localmente los mismos scripts y hacer las pruebas requeridas.

Lo primero es clonar el repositorio en el *Branch* donde ubiquemos el archivo *Singularity* que haga referencia a crear un contenedor, ya que si clonamos el archivo *Singularity* que se encuentra en el *máster* del repositorio, en algunos casos tiene internamente un documento HTML que no es legible para *Singularity*.

```
git clone -b <branch. ej: ...
    ipc-2018-seq-sat> <url. repo>
cd Singularity
sudo singularity build ...
    planner.img Singularity
```

Podemos clonar el repositorio de los dominios y problemas con el mismo comando git clone hacia el branch máster en la dirección *https://bitbucket.org/ipc2018-classical/domains/src/master/*, y buscar que problema y que dominio queremos aplicar al planificador. En este caso, todos los planificadores se ejecutaron con el problema 1(OPT o SAT dependiendo el caso del planificador) y el dominio correspondiente al directorio agrícola.

Luego de elegir el dominio y problema, creamos un directorio llamado rundir en el mismo directorio donde generamos el *build* del planificador para fines prácticos, el cual contendrá el dominio y el problema a planificar.

```
mkdir rundir
cp path/to/domain.pddl rundir
cp path/to/problem.pddl rundir
```

Terminando esto, ejecutamos los comandos que obtendrán la ruta en la que estamos trabajando, el dominio, el problema, el archivo de salida generado al finalizar la ejecución y los límites para poder ingresarlos de una manera mas fácil a la ejecución del planificador con singularity. El *Costbound* solamente se aplica a planificadores *cost-bounded track*.

```
RUNDIR="$(pwd)/rundir"
DOMAIN="$RUNDIR/domain.pddl"
PROBLEM="$RUNDIR/problem.pddl"
PLANFILE="$RUNDIR/sas_plan"
COSTBOUND=42
ulimit -t 1800
ulimit -v 8388608
singularity run -C -H $RUNDIR ...
    planner.img $DOMAIN ...
    $PROBLEM $PLANFILE $COSTBOUND
```

Luego de esto, esperamos a que se ejecute el proceso y analizamos los resultados.

5. Conclusión

Los planificadores al igual que cualquier desarrollo, se basa en diferentes algoritmos que pueden llevar a los mismos resultados, siendo estos en algunos casos más óptimos que otros en calidad de respuesta, tiempo requerido para la ejecución o recursos consumidos. En este caso se han probado algorimos de las mismas ramas *Optimal Tracks* y *Satisficing Tracks* para poder tener algunas comparativas de los diferentes algoritmos al realizar la misma acción. Por tomar un ejemplo, en el caso de *Scorpion* el algoritmo se completó en aproximadamente 30 minutos, mientras que *Complementary2*, realizando la misma tarea con los mismos datos, se completó en un tiempo aproximado de 5 minutos. Esto demuestra que la optimización de algoritmos en base a funciones asintóticas es muy importante cuando se trabaja con múltiples procesos o con múltiples datos, en donde los problemas cada vez se vuelven mas complejos.

Referencias

Francés Guillem, L. N. R. M., Geffner Hector. (2018). *Best first width search. complete, simulated and polynomial variants.* University of Basel, ICREA U Pompeu Fabra, University of Melbourne.

Franco Santiago, B. M., Levi H. S. Lelis. (2018). *The complementary 2 planner in the ipc.* School of Computing Engineering, University of Huddersfield, UK.

Guevara Patiño, R. (2016). *El estado del arte en la investigación.* Universidad Pedagógica Nacional, Bogotá, Colombia.

Guzmán Luna, J. A. (2003). *Técnicas de planificación para la generación automática de programas de control.* Universidad Nacional de Medellín, Colombia.

Seipp, J. (2018). *Scorpion planning.* University of Basel, Switzerland.

Seipp, R. G., Jendrik. (2018). *Fast downward stone soup.* University of Basel, Switzerland.

Speck David, M. R., Geiber Florian. (2018). *Symbolic planning based on evmdds.* University of Freiburg, Germany.