УДК 519.713

С. Л. ХАРЧЕНКО, ст. препод., ХНУРЭ, Харьков,

ЯЗЫК ПРОЕКТИРОВАНИЯ МОДЕЛИ ПОВЕДЕНИЯ СЛОЖНОЙ СИСТЕМЫ УПРАВЛЕНИЯ

В статье рассмотрены вопросы создания модели поведения сложной системы управления на основе применения формального языка проектирования. Это обеспечило визуализацию используемого параллелизма в модели поведения системы управления и показало возможные последовательности действий (трассы) при исполнении. Полученная модель поведения может быть использована при выполнении верификации и валидации управляющей программы системы.

Ключевые слова: модель поведения, процесс, асинхронные параллельные процессы, многопроцессорные системы управления, граф модели поведения.

Введение. Современные требования к системам управления ставят перед разработчиками всё более сложные задачи. Это проявляется в том, что системы управления должны обладать реакцией на динамически изменяющуюся внешнюю среду с учетом внутренних состояний, должны быть соблюдены принципы компонентного проектирования как аппаратной части системы так и специализированного программного обеспечения. В случае использования многопроцессорных систем должно быть реализовано динамическое взаимодействие компонент системы на принципах организации работы асинхронного параллельного или последовательно-параллельного процессов с использованием общих ресурсов системы.

Постановка задачи. Одной из актуальных задач разработки программного обеспечения - есть создание управляющих программ для систем. Из общего перечня работ, выполняемых при решении такой задачи, на ранних этапах проектирования, можно выделить задачу создания модели поведения будущей системы управления. Показатели качества ввыполнения такой работы могут быть существенно повышены, если при проектировании модели управления применить формальный язык, который позволит визуализировать трассы исполнения, организацию и взаимодействие параллельных процессов, при использовании многопроцессорной системы.

Поэтому, одним из основных требований к языку проектирования модели поведения сложной системы управления будет то, что конструкции языка проектирования модели поведения должны визуализировать трассы исполнения в системе управления, визуализировать возможные варианты параллелизма исполнения и быть пригодными для машинной обработки. Кроме этих требований в языке должна быть реализована возможность применения компонентного подхода, позволяющего поэтапно декомпозировать сложный объект до неделимых элементов или до элементов, находящихся в библиотеке системы проектирования.

Наличие такой модели, на ранних этапах проектирования, позволит выполнить проверку возможности достижения проектируемой системой поставленных целей при соблюдении условий определённых заказчиком.

Процесс в модели поведения. Если мы граф модели поведения обозначим как

P, и в этом графе ребра графа будут обозначать действия, а вершины состояния. В этом случае функционирование модели поведения можно рассматривать как переходы системы по действиям из одного состояния (s_i) в другое. В общем случае модель поведения может быть представлена как множество действий ***Act(P),*** которые может выполнить процесс ***P.*** Таким образом, для любого процесса

$$Act(P) \subseteq Act$$

где ***Act*** множество всех возможных действий.

Выбор множества действий модели поведения зависит от ситуации, для которой некоторое множество действий характерно. Если разделить множество действий на входные **α?**, выходные **α!** и внутренние **τ**, т.е. которые не связанны с внешней средой, то можно обозначить бесконечное множество имен объектов процесса (***Names)***, которые могут быть введены или выведены. Тогда множество действий можно определить как

$$Act = \{ \alpha? \mid \alpha \in Names\} \cup \{ \alpha! \mid \alpha \in Names\} \cup \{ \tau\}$$

В этом случае процесс в модели поведения может быть определён как

$$P = (S, s^0, R)$$

где ***S*** – множество состояний ***P;***

s^0 – начальное состояние ***P,*** $s^0 \in S$;

R – переходы вида (s_1, α, s_2), а $R \subseteq S \times Act \times S$ подмножество.

А сам процесс ***P***, в модели поведения, можно рассматривать как прохождение множества переходов вида $s^0 \xrightarrow{a0} s_1 \xrightarrow{a1} s_2$ … и выполнении действий ***a0, a1, ….*** Такой процесс будет продолжать работу до тех пор пока будет иметься переход из ***R*** и прекратит работу при отсутствии такового.

Если определим множество всех действий процесса ***P*** как ***Act(P)*** из ***Act\{τ}***, т.е.

$$Act(P) \overset{def}{=} \{ \alpha \in Act\backslash\{ \tau \} \mid^{\exists} (s_1 \xrightarrow{a} s_2) \in R \}$$

Процесс будет конечным, если множества ***S*** и ***R*** конечны.

Трассой процесса ***P*** будет конечная последовательность ***a0, a1, …*** множества ***Act,*** для которой имеется последовательность состояний $s_0, s_1, s_2, \ldots$ процесса ***P.*** Следует помнить, что s_0 совпадает с s^0 В этом случае множество всех трасс процесса ***P*** модели поведения можно обозначить как ***Tr(P)***.

Если из процесса модели поведения удалить все недостижимые состояния и все переходы, в которых присутствуют недостижимые состояния, то получившийся процесс ***P′*** (достижимая часть процесса ***P***) будет иметь такое же поведение, как и процесс ***P***, поэтому их можно рассматривать как одинаковые.

При работе с моделью поведения возможны замены состояний в процессе ***P.*** Если $s \in S$ мы заменим на произвольное $s' \in S$ и обозначим процесс как ***P′,*** который получается из ***P*** путем замены ***s*** на ***s′*** в множествах ***S*** и ***R***, то переход вида $s \xrightarrow{a} s_1$ можно заменить на переход вида $s' \xrightarrow{a} s_1$. В этом случае процесс ***P′*** будет обладать таким же поведением что и ***P.*** Если произвести множественную замену состояний в процессе ***P***, и это рассматривать как замену подмножества состояний процесса, то отображение этого процесса можно представить как $f : S \rightarrow S'$ и результатом будет

$$P' = (S', (s')^0, R')$$

где $(s')^0 \overset{def}{=} f(s^0)$;

и для каждой пары $s_1, s_2 \in S$ и $\alpha \in Act$

$$(s_1 \xrightarrow{a} s_2) \in R \Leftrightarrow (f(s_1) \xrightarrow{a} f(s_2)) \in R'.'$$

В этом случае поведение процессов одинаковое, следовательно, процессы одинаковы.

Операции, которые можно выполнять над процессом в модели поведения. При рассмотрении процесса моделирования можно прийти к заключению, что над ним может быть выполнено ограниченное количество действий. Теперь рассмотрим те действия, которые могут быть реализованы в модели.

Рассмотрим случай добавления к процессу **P** префиксного действия. В этом случае к множеству состояний процесса добавляется состояние **s**, которое является начальным состоянием нового процесса. Это приводит к тому, что к множеству переходов добавляется переход $\mathbf{s} \xrightarrow{a} \mathbf{s^0}$, а получившийся в этом случае процесс можно назвать как **α.P**.

Если имеется альтернативная композиция, предполагается то, что паре процессов $\mathbf{P_1}$ и $\mathbf{P_2}$ необходимо построить процесс **P**, который будет функционировать также как $\mathbf{P_1}$ или $\mathbf{P_2}$. Выбор, какую ветвь будет использовать процесс **P**, зависит от выбора самого процесса или от выбора внешней среды.

Так, если $\mathbf{P_1} = \alpha?.\mathbf{P_1}'$ и $\mathbf{P_2} = \beta?.\mathbf{P_2}'$ и окружающая среда может ввести **α,** но не может ввести **β**, то **P** должен выбрать единственно возможное поведение - $\mathbf{P_1}$**.** После этого процесс **P** не может изменить своего решения. Тогда альтернативная композиция выглядит следующим образом

$$P_i = (S_i, s_i^0, R_i)\ (i = 1,2)$$

и множество состояний S_1 и S_2 в этом случае не имеют общих элементов.

В этом случае альтернативной композицией будет

$$P_1 + P_2 = (S, s^0, R)$$

где: - S есть $S_1 \cup S_2$**,** к которому добавляется новое состояние s^0 которое будет начальным для $P_1 + P_2$;

- R содержит все переходы из R_1 и R_2;
- для каждого перехода из R_i $(i = 1,2)$ вида $\mathbf{s^0_i} \xrightarrow{a} \mathbf{s}$, R содержит переход $\mathbf{s^0} \xrightarrow{a} \mathbf{s}$.

Если множества S_1 и S_2 имеют общие элементы, то для процесса $P_1 + P_2$ необходимо заменить в S_2 те элементы, которые входят в S_1 и соответствующим образом выполнить модификацию R_2 и $s^0{}_2$**.**

Так как модель поведения является сложной системой, то в ней могут использоваться и параллельные композиции из нескольких взаимодействующих компонент. Если рассматривать две системы Sys_1 и Sys_2 (подсистемы), которые являются компонентами одной системы Sys**,** т.е.

$$Sys \overset{def}{=} \{ Sys_1, Sys_2 \},$$

то поведение систем может быть представлено процессами P_1 и P_2 соответственно, а поведение Sys_i $(i = 1,2)$ в составе системы Sys будет представлено соответствующим P_i**.** Обозначим $\{P_1, P_2\}$ как процесс, который описывает поведение системы. Это можно трактовать как обход графа P, при этом будем рассматривать все переходы из состояния в состояние в графе как мгновенные. Факт выполнения действия, в этом случае, будем фиксировать в момент перехода.

Каждое входное или выходное действие P_i $(i = 1,2)$ представляет собой результат взаимодействия P_i с процессом, не входящим в совокупность $\{P_1, P_2\}$**,** либо как результат взаимодействия с P_j, где $j \in \{1,2\}\backslash\{i\}$, либо это внутренне

действие процесса. Если это внутренне действие процесса то P_i $(i = 1,2)$ передаёт P_j $(j \in \{1,2\} \setminus \{i\})$ некий объект, а P_j его принимает.

Каждому возможному варианту поведения процесса P_i $(i = 1,2)$ можно сопоставить нить (в трактовке ОС UNIX – это процесс, т.е. последовательность операторов), которую можно обозначить как σ_i. Это позволит определить вариант поведения процесса σ_i $(i = 1,2)$, для P_i в составе процессов $\{P_1, P\}$.

Если обозначить совокупность всех вариантов поведения процессов как $Beh\{P_1, P_2\}$, каждый из которых соответствует одному из вариантов функционирования $\{P_1, P_2\}$, то можно предположить, что процесс $\{P_1, P_2\}$ функционирует последовательно, т.е. что при любом варианте $\{P_1, P_2\}$ образуется линейная упорядоченная последовательность $tr = (act_1, act_2, \ldots)$ действий, которые упорядочена по времени выполнения. Если обозначить C как одну из последовательностей, то множество индексов элементов последовательности можно обозначить как $Ind(tr)$, а множество точек как $Points(C)$.

Последовательность tr является линеаризацией C, если существует отображение

$$Lin: Points(C) \rightarrow Ind(tr)$$

В этом случае описание процесса $\{P_1, P_2\}$ можно определить как построение P отвечающему условию

$$Tr(P) = \bigcup_{C \in Beh\{P_1, P_2\}} Lin(C)$$

В этом случае в процессе P представлены все линеаризации процессов P_1 и P_2 отвечающие любому их совместному поведению.

Если процессы P_1 и P_2 имеют вид $P_i = (S_i, s^0_i, R_i)$ $(i = 1,2)$, а tr является трассой (S, s^0, R) процесса, компоненты которого можно определить как

- $S \overset{def}{=} S_1 \times S_2 \overset{def}{=} \{(s_1,s_2) \mid s_1 \in S_1, s_2 \in S_2\}$

- $s^0 \overset{def}{=} (s^0_1, s^0_2)$

- для каждого перехода $s_1 \xrightarrow{\alpha} s'_1$ и каждого состояния $s \in S_2$ из R_1. R содержит переход $(s_1, s) \xrightarrow{\alpha} (s'_1, s)$. И для каждого перехода $s_2 \xrightarrow{\alpha} s'_2$ из R_2 и каждого состояния $s \in S_1$, R содержит переход $(s, s_2) \xrightarrow{\alpha} (s, s_2)$.

Для каждой пары переходов $s_1 \xrightarrow{\alpha} s'_1 \in R_1$ и $s_2 \xrightarrow{\alpha} s'_2 \in R_2$, R содержит переход $(s_1.s_2) \xrightarrow{\tau} (s'_1.s'_2)$.

Можно утверждать, что данный процесс P является параллельной композицией процессов P_1 и P_2, которую можно обозначить как $P_1|P_2$.

При работе с параллельными процессами применяются ограничения, которые вытекают из логики функционирования процессов P_1 и P_2. Если L произвольное подмножество множества $Names$, то в этом случае ограничение по L есть процесс $P\backslash L = (S, s^0, R')$ который получается путем удаления всех переходов имеющих метки с именами из L, т.е.

$$R' \overset{def}{=} \{(s \xrightarrow{\alpha} s') \in R \mid \alpha = \tau, \text{ или } name(\alpha) \in \not{L}\}$$

Следует помнить и о том, что при работе с параллельными композициями множество состояний процесса P определяется как произведение состояний процессов P_1 и P_2, входящих в этот процесс.

Так как модель поведения строится на компонентном принципе, это может приводить к многократному использованию одного и того же компонента в модели. Это приводит к тому, что повторяются метки переходов и имена действий. Для разрешения такого рода конфликтов выполняется переименование имен $f : Names \rightarrow Names$ и изменении меток **α?, α!** на ***f*(α)?** и ***f*(α)!**. Получившийся процесс можно обозначить как ***P[f]***.

При не тождественном переименовании, когда изменяются только имена из списка $\alpha_1, \ldots, \alpha_n$ и происходит их отображение в имена $\beta_1, \ldots, \beta_n$ В этом случае эквивалентное обозначение процесс имеет вид $P[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n]$ [1].

Трактовка конструкций графа модели поведения. Исходя из того, что мы рассматриваем модель поведения, а это подразумевает работу с такими категориями как состояние системы, действие, событие, переход из одного состояния в другое, условия перехода, последовательность событий, то из графа ***P*** можно выделить информационный (**J)** и управляющий (**G)** графы. Если сравнивать переходы в графах **J, G** и переходы в графе описания модели поведения, то можно получить утверждение об их трассовой эквивалентности $Tr(P) = Tr(\mathrm{G}) = Tr(\mathrm{J})$, что позволит оперировать каждым из графов при решении специфичных задач создания программного обеспечения.

Так к графу **J** можно отнести все действия, которые изменяют внутреннее состояние модели поведения. Теперь определим, что подразумевается под термином действие - любая операция, которая производит запись в порт ввода/вывода или выполняет отщепление нового процесса в системе. Следует помнить, что декларации присвоения необходимо относить к операциям изменения внутреннего состояния процесса.

Проведя анализ управляющего графа **G,** можно выделить несколько моментов. Так, переход из состояния в состояние может быть безусловным или может быть ограничен выполнением некоторого условия. Такое условие влияет на выбор трассы и зависит или от состояния объектов связанных с внутренними или входными действиями. Наиболее удобно условия определяющие выбор трассы записывать над ребром графа. Это существенно улучшит визуализацию условия перехода. Вид такой записи предполагает следующие варианты:

- отсутствие записи над ребром, безусловный переход;
- наличие записи над ребром, переход будет выполнен, если выражение над ребром истинно. Особо надо подчеркнуть, что в графе обязательно должна иметься альтернативная трасса, которая будет реализована в случае невыполнения условия;
- наличие над ребром символа «*» означает, что реализуется безусловное выполнение перехода (используется только в альтернативной конструкции).

Следующую ситуацию в управляющем графе необходимо рассматривать, когда используются многопроцессорные платформы, на которых реализуется параллельные асинхронные процессы. В этом случае необходимо рассматривать параллельные процессы как два или более компонента. Признаком организации параллельных процессов будут служить символы “**&&**>“, а условием завершения параллелизма будут служить символы “>**&&**“ или “>||“. Эти конструкции должны быть связаны с состоянием, т.е. с вершиной графа.

Так как условия ветвления и исполнения циклических операторов необходимо реализовывать с учетом выбора наиболее оптимальной конструкции, то на данный момент разработчику рекомендуется указывать наиболее приемлемый оператор

ветвления, например, “>**if/case**“, который ассоциирован с вершиной графа реализующей входные действия. При отсутствии такого указания интерпретатор будет реализовывать схему «**if**». Аналогичные правила относятся и к выбору оператора цикла. Если отсутствует явное указание выбирать тип цикла, например “>**for/while**“, то интерпретатор выберет реализацию конструкции “>**while**“.

Правила конструирования графа и выполнения надписей. Для удобства чтения граф конструируется слева на право и сверху вниз. Все элементы графа (вершины и рёбра) имеют уникальную маркировку. Так как из одной вершины (состояния) могут выходить несколько рёбер (действий), то порождающая вершина графа тиражируется с указанием признака тиража – «`» в идентификаторе вершины. Аналогичная конструкция возникает и в ситуации, когда несколько рёбер входят в вершину графа. При переносе элементов на «новую строку» идентификаторы вершин повторяются с указанием операции переноса «``».

Относительно надписей на графе процесса. Необходимо выделить несколько типов - это рекомендательные условия выполнения и действия. Их общей характеристикой будет то, что они характеризуются как поток символов, записанных по некоторым правилам. Теперь рассмотрим их по порядку.

Группа рекомендательных надписей относится только к «входным» вершинам графа (состояниям). Их характерной чертой является наличие парного символа «“», как ограничения потока символов. Это «входные» рекомендательные конструкции: “>**if“, “>case“, “>for“, “>while**“. Если они отсутствуют, как рекомендательные, то интерпретатор обязан ситуативно реализовывать стандартные конструкции if и while.

Существует ещё одна конструкция, которая относится к «входным» - запуск параллельных процессов “**&&**>“. Эта конструкция является управляющей и предполагает наличие нескольких параллельных процессов. В иных ситуациях использование этой конструкции запрещено. Признаком завершения параллельных процессов в модели поведения служат конструкции “>**&&**“ и “>||“, которые определяют механизм завершения параллельных процессов. По правилам записи эти конструкции ассоциированы с вершиной графа, для которой реализуется сценарий «переход».

Теперь рассмотрим такой элемент, как условие выполнения. Этот поток символов записывается над ребром графа и предполагает возможность записи любого вида логического выражения, в том числе и составного. Результатом выполнения логического выражения есть получение заключения - «истинно» или «ложь». Такая конструкция наиболее применима в реализации циклического оператора **while** и оператора ветвления **if.**

Но в ситуации, когда используется конструкция **case** необходимо значение переменной сравнивать с шаблоном и по результату сравнения принимается решение о возможности выполнения действия ассоциированного с ребром графа. Так как в этом случае в условии должен находиться «шаблон», то значение переменной, с которым производится сравнение, должно быть определено в операторе case, поэтому полная запись рекомендованной конструкции должна быть представлена как **“>case“$<имя переменной>**. Между конструкциями **“>case“** и **$<имя переменной>** допускаются пробелы, которые игнорируются при разборе строки символов. При записи самих шаблонов необходимо использовать конструкцию **“$<имя переменной>**, причем последним шаблоном в операторе

всегда должен быть шаблон **$*.** Его исполнение означает отсутствие совпадения всех предыдущих шаблонов со значением, которое хранится в переменной ассоциированной с заданием условия выполнения конструкции **case.**

Рассматривая конструкцию условия для ситуации **“>for“**, необходимо отметить, что в этом случае задается диапазон изменения значения переменной и правило, по которому циклически изменяется её значение. Поэтому в условии, которое записывается над ребром графа, должна формироваться последовательность групп символов разделенных пробелами или другими специальными символами. Пример формирования такого потока символов:

$<имя переменной> = <минимальное значение переменной (константа)> $<имя переменной> = $<имя переменной> <тип операции> {$<имя переменной> или <константа>} <максимальное значение переменной (константа)>

Все элементы текстовой конструкции должны быть разделены пробелами или специальными символами. Следует обратить внимание на такой элемент текстовой конструкции как **<тип операции>**. В этом случае должны рассматриваться только арифметически операции. Для конструкции **$<имя переменной> = $<имя переменной> <тип операции> {$<имя переменной> или <константа>}** можно использовать и унарные операции с построением соответствующей конструкции.

Следует отметить и особенность использования имен переменных, как в элементах «условие», так и в элементах «действие». Общим признаком имени переменной является символ **«$»**, находящийся в первой позиции последовательности символов.

Выводы. Проектирование модели поведения системы управления – это первый шаг к созданию системы управления. И если при выполнении этой работы можно сократить количество ошибок «человеческого фактора», то это может произойти только при внедрении в процесс проектирования формального языка с высокой степенью визуализации.

Так как формальная запись созданной модели поведения имеет все признаки скриптового языка программирования, то очевидно, что такое представление модели поведения может быть использовано, на ранних этапах проектирования, для проведения проверки возможности достижения создаваемой системой поставленных целей, при соблюдении условий, которые сформулированы заказчиком.

Получение положительного заключения позволит приступить к следующей фазе реализации проекта – генерации кода управляющей программы по формальному описанию модели поведения системы управления.

Список литературы: 1. А. *М. Миронов* Теория процессов /Internet ресурс/ http://www.twirpx.com/file/617525/

Поступила в редколлегию 20.06.2013

УДК 519.713

Язык проектирования модели поведения сложной системы управления/ Харченко С. Л. // Вісник НТУ «ХПІ». Серія: Нові рішення в сучасних технологіях. – Х: НТУ «ХПІ», – 2013. - № 56 (1029). – С.37-44 . – Бібліогр.: 1 назв.

У статті розглянуто питання створення моделі поведінки складної системи керування на засадах формальної мови проектування. Це забезпечило візуалізацію використовуваного паралелізму у моделі поведінки системи керування і показало можливі послідовності дій (траси) при виконанні. Отримана модель поведінки може бути використана при виконанні верифікації та валідації керуючої програми системи.

Ключові слова: модель поведінки, процес, асинхронні паралельні процеси, багатопроцесорні системи управління, граф моделі поведінки.

The paper deals with creating a complex pattern of behavior management system based on the use of the formal language of design. Which provided visualization used in the model of concurrency control system behavior and the possibility of action sequences (the track) with the performance. The resulting pattern of behavior may be useful in the verification and validation of the control program of the system.

Keywords: pattern of behavior, a process, asynchronous parallel processes, multi-processor control system, the graph model of behavior.

УДК 004.73

К. В. КОЛЕСНІКОВ, канд. техн. наук, доц., ЧДТУ, Черкаси;
А. Р. КАРАПЕТЯН, аспірант, ЧДТУ, Черкаси;
Т. А. ЦАРЕНКО, студент, ЧДТУ, Черкаси

ГЕНЕТИЧНІ АЛГОРИТМИ ДЛЯ ЗАДАЧ БАГАТОКРИТЕРІАЛЬНОЇ ОПТИМІЗАЦІЇ В МЕРЕЖАХ АДАПТИВНОЇ МАРШРУТИЗАЦІЇ ДАНИХ

Представлені існуючі підходи і методи застосування генетичних алгоритмів для рішення задач багатокритеріальної оптимізації. Розглянуто можливість формалізації багатокритеріальної задачі пошуку оптимальних шляхів у корпоративній комп'ютерній мережі. Проаналізовано обчислювальну складність генетичного алгоритму пошуку оптимальних шляхів на графі.

Ключові слова: маршрутизація, генетичний алгоритм, багатокритеріальна оптимізація, Парето-оптимальність, метод рангів Голдберга, гібридний генетичний алгоритм HAG, пошук оптимального шляху на графі, хромосоми, кросовер, мутація, відбір.

Вступ. Однією з найпоширеніших функціональних задач мережі є задача про знаходження найкоротшого маршруту, а саме пошук шляху між двома визначеними вершинами графа, який відповідає найменшому значенню певного функціонала за визначеним критерієм. Ця задача застосовується у сферах транспорту, маршрутизації, комунікації. Існує низка класичних алгоритмів розв'язання цієї задачі (Беллмана-Форда, Дейкстри, Флойда-Уоршелла, Джонсона) [1,2]. Для зменшення часу виконання цих алгоритмів сформовано ряд їх паралельних версій, що дозволяють підвищити продуктивність роботи поширених алгоритмів маршрутизації.

Для розв'язання задач маршрутизації використовуються класичні алгоритми, переважна кількість яких оперують лише одним параметром оптимізації – вагою (ціною) шляху, що виражає сукупність його адитивних характеристик. Проте, зазвичай, існує декілька параметрів, які характеризують кожну гілку мережі, наприклад, пропускна здатність, затримка, швидкість передачі, навантаження, надійність, які можна поділити на адитивні та неадитивні. Таким чином, у сучасних мережах з'явилась необхідність розв'язання задачі про найкоротші шляхи з кількома критеріями оптимізації. Обчислювальні затрати на розв'язання таких задач експоненційно зростають із ростом розмірності оброблюваних графів. Тому виникає актуальна необхідність формування нових підходів та алгоритмів розв'язання задач пошуку оптимальних шляхів з багатьма критеріями, одним із яких є метод генетичних алгоритмів [3].