ВЕСТНИК
ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
МОРСКОГО И РЕЧНОГО ФЛОТА ИМЕНИ АДМИРАЛА С. О. МАКАРОВА

ЛОГИСТИКА И ТРАНСПОРТНЫЕ ТЕХНОЛОГИИ

УДК 519.85:656.62.022.5

В. А. Стальмаков,
аспирант,
ГУМРФ имени адмирала С. О. Макарова

ПАРАЛЛЕЛЬНЫЙ ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ СОСТАВЛЕНИЯ РАСПИСАНИЯ ПРОХОЖДЕНИЯ СУДОВ ЧЕРЕЗ ШЛЮЗОВАННЫЕ СИСТЕМЫ И ЕГО ВЕРИФИКАЦИЯ

PARALLEL GENETIC ALGORITHM FOR SOLVING SCHEDULING PASSING THROUGH THE GATEWAYS SYSTEM AND ITS VERIFICATION

В статье рассматривается алгоритм составления расписания прохождения судов через шлюзованные системы, основанный на использовании генетических алгоритмов и распределенных вычислений. Производится его верификация на модели.

This article discusses an algorithm for scheduling the passage of ships through the sluice system based on genetic algorithms and distributed computing. Made it to the verification of the model using temporal logic.

Ключевые слова: генетический алгоритм, верификация на модели, темпоральная логика.
Key words: genetic algorithm, model verification, temporal logic.

Введение

В современной России эксплуатируются десятки тысяч километров внутренних водных путей, на которых функционирует более 150 шлюзов. Эффективность эксплуатации флота на водных путях в значительной степени зависит от времени прохождения через судоходные гидротехнические сооружения, а также от стабильности и надежности работы судоходных шлюзов.

Известно, что шлюзование является одним из наиболее эффективных способов улучшения судоходных условий. Однако опыт эксплуатации показывает, что шлюзование имеет и негативную сторону: судоходные шлюзы являются своего рода тормозом транспортного процесса. Более того, вследствие характерной для речного флота неравномерности распределения грузо- и пассажиропотоков (как в течение суток, так и всей навигации) нередко происходит скопление в бьефах судов и составов, ожидающих шлюзования. Очевидно, что от качества расписания шлюзований зависит эффективность эксплуатации шлюзованной системы. Поэтому составление оптимального расписания шлюзований является важной задачей для повышения эффективности эксплуатации водного транспорта.

Задача составления расписания прохождения судов через шлюзованные системы

Рассмотрим задачу составления расписания прохождения судов через шлюзованные системы. Пусть шлюзованная система (ШС) двухниточная, состоит из однокамерных шлюзов (на примере шлюзов № 7 и 8 Шекснинского гидроузла); время подхода к подходному каналу ШС и состав групп судов, проходящих через данную ШС, известны заранее; также известно минимальное среднее время шлюзования. Тогда требуется составить расписание прохождения групп судов через данную ШС (план-график шлюзований) таким образом, чтобы все группы судов были включены в расписание; не возникало накладок в расписании; время ожидания шлюзования групп судов в подходном канале было минимальным; время простоя ШС было минимальным; соблюдалась

последовательность пропуска судов, определенная в Правилах пропуска судов и составов через шлюзы внутренних водных путей РФ.

Математическая модель

Исходные данные

В качестве исходных данных для составления расписания выступают множества:

— $G = \left\{ g_1, g_2, g_3, \ldots, g_{N_g} \right\}$ — нитей шлюза;

— $S = \left\{ s_1, s_2, s_3, \ldots, s_{N_s} \right\}$ —судов;

— $T = \left\{ t_1, t_2, t_3, \ldots, t_{N_t} \right\}$ — временны́х интервалов, описывающих режим работы ШС,

где N_g — количество нитей шлюза, N_s — количество судов, N_t — количество временны́х интервалов.

Расписание прохождения групп судов через ШС можно определить векторами (γ, τ):

$$\gamma = \left(\gamma_1, \ldots, \gamma_i, \ldots, \gamma_{N_g} \right), \tau = \left(\tau_1, \ldots, \tau_i, \ldots, \tau_{N_t} \right), \quad (1)$$

где $\gamma_i \in G$ — код нити шлюза, назначенной группе судов для прохождения ШС; $\tau_i \in T$ — код временно́го интервала, назначенного группе судов для прохождения ШС.

Описание ограничений

Все требования, предъявляемые к расписанию прохождения судов через ШС, разобьем на обязательные и желательные. К обязательным относятся требования, невыполнение которых делает невозможным осуществление процесса пропуска судов через ШС. Эти требования будем рассматривать в качестве ограничений. К желательным относятся, например: минимизация времени стоянки судов в ожидании шлюзования и т. д. Эти требования будем рассматривать в качестве критериев.

Отсутствие накладок шлюзования групп судов

$$\forall \left(g_i, t_k \right) : g_i \in G, t_j \in T \left(\exists ! g_i : \left(g_i = g_r \right) \wedge \left(g_i \in Z^{t_j} \right) \right) \vee \left(\neg \exists g_i : \left(g_i = g_r \right) \wedge \left(g_i \in Z^{t_j} \right) \right), \quad (2)$$

где Z^{t_j} — множество шлюзований, происходящих во временно́й интервал t_j.

Критерий качества расписания

Для оценки полученного расписания введем критерий P качества расписания

$$P = f(\tau) = \sum_{i=1}^{N} c_i w_i(\tau), \quad (3)$$

где c_i — коэффициент штрафа за невыполнение i-го требования; w_i — оценка, определяющая степень выполнения i-го требования. Чем больше значение критерия P, тем хуже расписание.

Теперь сформулируем задачу составления расписания. Для заданных множеств G, S и T требуется найти такое расписание, определенное вектором (1), содержащим коды временны́х интервалов, назначенных группам судов, которое удовлетворяет ограничениям и минимизирует значение критерия потерь качества расписания:

$$P \to \min. \quad (4)$$

Параллельный генетический алгоритм

Задача составления расписания прохождения судов через шлюзованные системы относится к классу NP — полных задач, сложность решения которых растет экспоненциально с ростом числа и возможных значений изменяемых переменных. Поэтому для решения таких задач целесообраз-

но использовать эвристические методы. Одними из самых популярных и перспективных эвристических алгоритмов на сегодняшний день являются генетические алгоритмы.

Генетический алгоритм — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и скрещивание.

Структура особи

В генетическом алгоритме решения задачи составления расписания прохождения групп судов через ШС каждая особь является одним из возможных решений задачи, т. е. вариантом расписания. Формула (1), как было сказано, выражает математическую модель расписания. Следуя этой модели, предлагается рассматривать особь, состоящую из одной хромосомы (рис. 2).

Хромосома в свою очередь состоит из генов, обозначаемых целыми числами 1, 2, 3, …, *i*, …, *N*, причем номер гена соответствует номеру группы судов, так, *i*-й ген характеризует группу судов g_i из множества *G*.

Информационным наполнением хромосомы является время прохода группы судов через ШС. Таким образом, значением *i*-го гена является номер (код) временно́го интервала из подмножества допустимых временны́х интервалов, в который предполагается начать проведение группы судов через ШС. Это означает, что хромосома связана с блоком групп судов особой связью, которую можно назвать связью «однозначного соответствия».

Создание начальной популяции

Формирование каждого варианта расписания происходит следующим образом. Каждому гену хромосомы, условно обозначающему группу судов, приписывается некоторое случайное значение — номер временно́го интервала из подмножества допустимых для данной группы судов временны́х интервалов. Аналогичным образом формируются следующие варианты расписания.

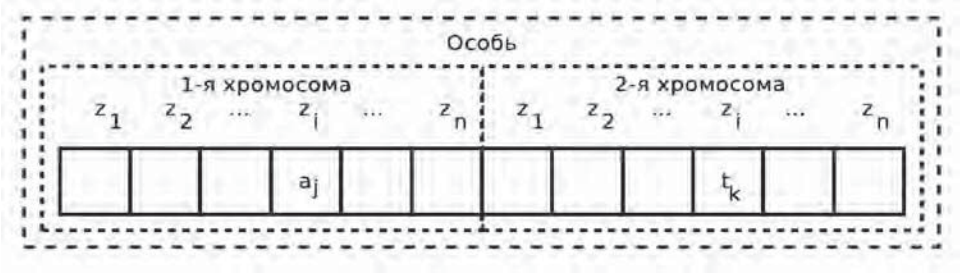

Рис. 1. Структура особи

Отбор

На данном шаге происходит отбор наиболее приемлемых вариантов расписания, имеющих наиболее предпочтительные значения используемой оценочной функции по сравнению с остальными вариантами. В данной работе предлагается метод, называемый «элитным отбором» или «элитной стратегией», который при решении данной задачи заключается в следующем: из предыдущей популяции выбирается только некоторое число отдельных вариантов расписания, имеющих наименьшее значение весовой функции — критерия, отражающего выполнение желательных требований. Выявленные таким способом «элитные» варианты без каких-либо изменений переходят в следующее поколение. Оставшееся количество «свободных мест» в новой популяции заполняется вариантами, полученными в результате их скрещивания и мутации (рис. 3).

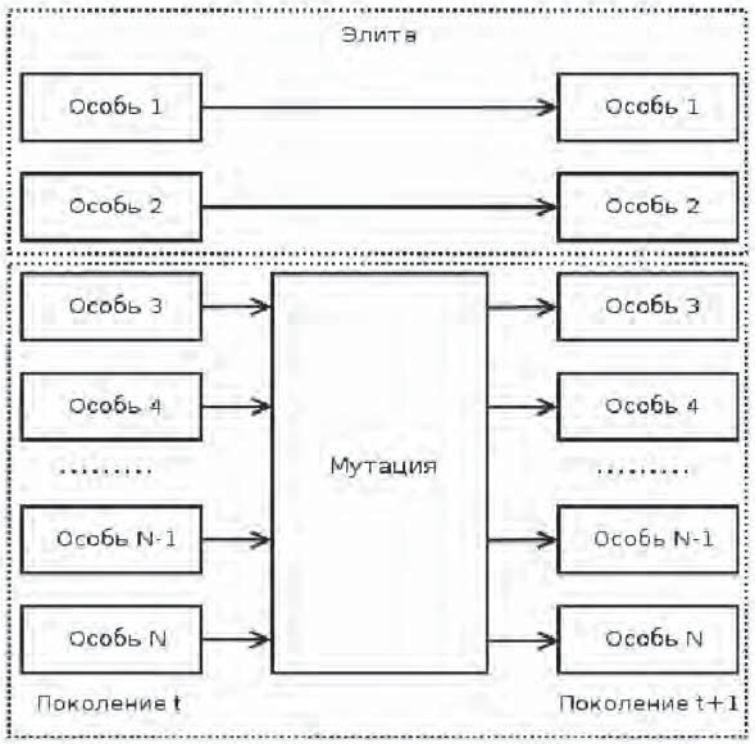

Рис. 2. Схема отбора

Скрещивание

В предлагаемом генетическом алгоритме используется одноточечное скрещивание. Оно происходит по следующей схеме: случайным образом выбираются *n* пар вариантов расписания из текущего поколения. Далее для каждой пары отобранных вариантов случайным образом разыгрывается позиция гена (локус) и производится обмен участками генетического кода между вариантами (рис. 4). Таким образом получается новая пара вариантов, которая помещается в новое поколение.

Рис. 3. Скрещивание

Мутация

При формировании нового поколения к вариантам расписания предыдущего поколения, не относящимся к «элитным», применяется операция мутации.

В данной задаче реализованы два вида мутации:

1) случайным образом выбирается одна группа судов. Если для нее доступны другие временны́е интервалы, в которые не нарушается ее расписание, то шлюзование данной группы судов переносится на один из этих интервалов, выбранный случайным образом;

2) случайным образом выбираются два шлюзования, которые проводятся для судов одного типа. Если возможно, то эти шлюзования меняются местами.

Выбор вида мутации производится случайным образом.

Критерий останова и выбор «наилучшей» особи

В результате применения операторов отбора и мутации формируется популяция потомков, которая заменяет родительскую популяцию, после чего выполняется проверка условия останова алгоритма. Если заданное значение максимального количества итераций алгоритма достигнуто, то алгоритм завершает работу. В последнем поколении в качестве решения задачи выбирается тот вариант расписания, который в наибольшей степени удовлетворяет требованиям, предъявляемым к расписанию. То есть особь, имеющая минимальное значение оценочной функции.

Известно, что генетические алгоритмы хорошо поддаются распараллеливанию. Для них предложено множество архитектур и стратегий поиска для выполнения на параллельных процессорах. Данные стратегии основаны на взаимодействии изолированных популяций или отдельных генетических алгоритмов, вычисление решений которых реализуется на отдельных процессорах. Взаимодействие популяций или генетических алгоритмов реализуется по специальным правилам, формирующим суть стратегии и требующим специальной параллельной архитектуры процессоров. Однако в настоящий момент широко распространены и общедоступны многоядерные процессоры (2-, 3- и 4- ядерные), где каждое ядро представляет собой полноценный процессор. Данную систему можно рассматривать как простейшую многопроцессорную систему. Рассмотрим несколько подходов к распараллеливанию генетических алгоритмов.

Первым подходом является распараллеливание отдельных шагов алгоритма: отбора, скрещивания, мутации, оценки особей. При этом генофонд разделяется на блоки, над каждым из которых работает отдельный поток. Однако такой подход не очень удобен в программировании по сравнению с двумя другими.

При втором подходе, его еще называют островной моделью, заводят несколько независимых генофондов (островов), каждый из которых итерируют в отдельном потоке. Затем каждые несколько итераций устраивают «миграцию» — переселение ДНК между всеми «островами». Стратегию переселения можно выбирать свободно: можно переселять чемпиона, можно — случайную ДНК (или несколько ДНК). Однако стоит помнить, что слишком частые миграции, а также миграции большого числа ДНК со временем нивелируют популяции на разных «островах».

Такой подход имеет неоспоримые преимущества. Во-первых, его легко воплотить программно, так как не требуются затраты на синхронизацию потоков. Во-вторых, на каждом «острове» можно использовать различные стратегии мутации, размножения и т. д., что приведет к более широкому охвату области возможных значений переменных и с большей вероятностью к глобальному экстремуму, нежели к локальному.

Третий подход является модификацией островной модели из расчета на массовый параллелизм. Его называют клеточной моделью. Такой подход можно использовать, когда речь идет о вычислениях в кластере. Суть клеточной модели в следующем: пусть у нас имеется множество процессоров (машин), расположим их логически в виде таблицы.

Каждая ячейка таблицы представляет собой один процессор. Каждый процессор итерирует отдельную популяцию, со своим набором стратегий размножения, мутации и т. д. По окончании определенного числа итераций происходят миграции между соседними ячейками таблицы.

Такой подход применим лишь на многоядерных машинах либо в кластерах, так как на машинах с малым числом ядер возникнут большие накладные расходы, связанные с переключением контекста потока, что резко снизит эффективность параллельных вычислений.

Преимуществом клеточной модели перед островной является резкое снижение вероятности скатывания в локальный минимум, так как перемешивание происходит не между всеми популяциями, как в островной модели, а лишь между соседними, что снижает скорость выравнивания генофонда между популяциями.

Для решения данной задачи предлагается алгоритм, являющийся некоей комбинацией островной и клеточной моделей.

В данном случае имеется несколько независимых популяций (островов), эволюция в которых происходит параллельно; и буфер обмена особями, который в каждый момент времени либо

содержит одну особь, либо пуст. Процесс миграции особей между островами использует этот буфер обмена. Процесс миграции запускается при достижении на одном из островов критерия необходимости миграции.

Существует *n* независимых популяций (островов), эволюция в которых происходит параллельно, и буфер обмена особями. Эволюция на каждом острове может находиться в трех состояниях:

1) создание нового поколения (основной процесс эволюции);

2) пауза (проверка буфера обмена);

3) достигнут критерий необходимости миграции (ожидание миграции).

Буфер обмена может находиться в состояниях:

1) пуст;

2) содержит особь из острова, в котором достигнут критерий необходимости миграции;

3) остров для миграции выбран, происходит обмен особями.

После каждого создания нового поколения проверяется условие достижения критерия необходимости миграции, и если он не выполняется, то эволюция переходит в состояние паузы. В этом состоянии проверяется буфер обмена особями. Если он содержит особь, предназначенную для миграции (с исходящего острова), то она принимается в популяцию взамен наихудшей особи, а копия наилучшей особи популяции помещается в буфер обмена для перехода в исходящий остров.

Если же условие критерия необходимости миграции выполняется, то в буфер обмена (при условии, что он свободен) помещается наилучшая особь популяции или ожидается освобождение буфера обмена. Затем ожидается переход буфера обмена в состояние обмена особями и в популяцию из него принимается особь вместо наихудшей. Буфер обмена освобождается, процесс эволюции переходит в состояние создания нового поколения.

Таким образом миграция особей происходит между двумя островами и только при условии выполнения критерия необходимости миграции.

Верификация предложенного алгоритма

Написание любых программных продуктов обычно сопровождается различными подходами по определению корректности приложения. Наиболее распространенным методом проверки корректности приложения является тестирование. Однако, несмотря на относительную простоту написания тестовых сценариев и проверки дефектов программы на этих сценариях, разработчик никак не сможет доказательно удостовериться в корректности выполнения программы с помощью этого подхода.

Другими подходами являются методы спецификации контрактов и их проверка в статическом либо динамическом режиме, а также верификация программной модели.

Верификация программной модели позволяет доказательно проверить корректное поведение на всех возможных входных воздействиях.

Виды верификации

Формальная верификация

Формальная верификация представляет собой процесс доказательства с помощью формальных методов корректности или некорректности алгоритмов, программ и систем в соответствии с заданным описанием их свойств. Она требует высококвалифицированных специалистов в области формальных доказательств и логического вывода.

В общем случае задача, решаемая в рамках данного подхода, является алгоритмически неразрешимой. При этом весь процесс формального доказательства связан с огромной ручной работой, что делает его малоприменимым на практике.

Верификация на модели

Под верификацией на модели понимают метод формальной верификации, позволяющий проверить, удовлетворяет ли заданная модель системы спецификациям, написанным на формаль-

ном языке. Применение данного подхода позволяет для заданной модели поведения системы с конечным (возможно, очень большим) числом состояний проверить выполнимость некоторого логического требования (спецификации), обычно формулируемого в терминах языка темпоральной логики (LTL, CTL и т. д.). Таким образом можно проверить не только условия на мгновенное состояние системы, но и историю его развития со временем.

Проверка предложенного алгоритма

Для проверки корректности предложенного алгоритма необходимо:

1) сформулировать требования, предъявляемые к алгоритму;

2) выбрать логический язык, на котором можно описывать требования, предъявляемые к алгоритму, и представить эти требования в виде формул;

3) выбрать математическую модель, адекватно представляющую все вычисления алгоритма. Модель должна быть устроена так, чтобы каждое вычисление в модели являлось интерпретацией языка;

4) проверить корректность формул на всех вычислениях модели.

При верификации параллельных алгоритмов, как правило, требуется проверить, что в каждом вычислении системы некоторые события происходят в определенной последовательности. Каждое событие можно охарактеризовать булевой переменной (0-местным предикатом), которая принимает значение «Истина» тогда и только тогда, когда наступает событие. Таким образом, в языке не нужны предметные переменные, термы, кванторы. Однако осуществимость событий изменяется со временем. Значит, в логическом языке должен быть явно учтен фактор времени. То есть для описания требований, предъявляемых к параллельному алгоритму, достаточно воспользоваться языком пропозициональной темпоральной логики линейного времени PLTL.

В PLTL наряду с булевыми логическими связками для описания причинно-следственной зависимости событий во времени применяются темпоральные операторы:

— X — в следующий момент времени;

— F — когда-то в будущем;

— G — всегда в будущем;

— U — до тех пор, пока;

— R — освободить.

Пусть задано множество булевых переменных (атомарных высказываний) $AP = \{p_1, p_2, ..., p_n\}$, тогда формула PLTL — это p_i, если $p_i \in AP$. Если φ и ψ формулы PLTL, то будут справедливы следующие булевы высказывания:

— $(\varphi \vee \psi)$;

— $(\varphi \wedge \psi)$;

— $(\varphi \rightarrow \psi)$;

— $(\neg\varphi)$.

Также будут определены следующие выражения:

— $(X\varphi)$; — в следующий момент будет верно φ;

— $(F\varphi)$ — когда-то в будущем будет верно φ;

— $(G\varphi)$ — всегда верно φ;

— $(\varphi U \psi)$ — φ остается верной, пока не станет верной ψ;

— $(\varphi R \psi)$ — ψ может перестать быть верной, только после того как станет верной φ.

Чтобы облегчить запись формул и избавиться от лишних скобок, условимся, что одноместные темпоральные операторы X, F, G обладают таким же приоритетом, как отрицание ¬, а двухместные темпоральные операторы U, R имеют наивысший приоритет среди двухместных связок.

Введем следующие атомарные высказывания, соответствующие основным событиям вычисления алгоритма:

— run_i — i-й остров находится в процессе создания нового поколения;

— $pause_i$ — пауза между итерациями;

— $met\_criterion_i$ — достигнут критерий необходимости миграции;
— $exchange_i$ — обмен особями с буфером обмена;
— *has_ first* — в буфер обмена помещена первая особь для обмена;
— *has_ second* — в буфер обмена помещена вторая особь для обмена.

Теперь сформулируем требования к алгоритму и выразим их в виде формул PLTL.

Если остров достиг критерия необходимости миграции и буфер обмена свободен, то когда-нибудь он поместит в буфер обмена особь для миграции

$$G(\neg has\_\ first \wedge met\_criterion_i \rightarrow F\ has\_\ first).$$

Если остров перешел в состояние паузы и буфер обмена содержит первую особь и не содержит второй особи для обмена, то остров когда-нибудь поместит в буфер обмена особь

$$G(has\_\ first \wedge \neg has\_\ second \wedge pause_i \rightarrow \mathrm{F}\ has\_\ second).$$

В качестве системы, осуществляющей проверку на модели, была выбрана система NuSMV. Каждая программа, воспринимаемая системой NuSMV, представляет собой набор модулей, которые, в свою очередь, описывают свое внутреннее состояние, а также систему переходов в зависимости от входных воздействий.

В каждой программе существует модуль main, который указывает конфигурацию других модулей и является в некотором смысле стартовой точкой программы.

Каждый модуль имеет:
— аргументы, передаваемые ему при создании;
— набор внутренних переменных;
— блоки управления значением внутренних переменных.

Система позволяет использовать в качестве основных типов только примитивные целые (byte, short, int, boolean) массивы со статической длиной, а также типы перечисления.

Блоки управления значением внутренних переменных разделяются на две группы:
— блоки, отвечающие за инициализацию переменных (init);
— блоки, отвечающие за изменение переменных при различных условиях (next).

Условия задаются в виде булевой формулы, использующей обычные булевы операторы. В блоках изменения внутренних переменных (next), различных для условия описания, используется конструкция множественного условия (case).

Ниже приведен код модели данного алгоритма на языке NuSMV.

```
MODULE main
 DEFINE
  clipboard_has_first := (isl1.status = met_criterion | isl2.status = met_criterion | isl3.status = met_criterion | isl4.status = met_criterion );
  clipboard_has_second := (isl1.status = met_criterion | isl2.status = met_criterion | isl3.status = met_criterion | isl4.status = met_criterion ) & clipboard_has_first;
 VAR
  isl1                    : process island(clipboard_has_first, clipboard_has_second);
  isl2                    : process island(clipboard_has_first, clipboard_has_second);
  isl3                    : process island(clipboard_has_first, clipboard_has_second);
  isl4                    : process island(clipboard_has_first, clipboard_has_second);
LTLSPEC G (!(clipboard_has_first = FALSE & clipboard_has_second = TRUE));
LTLSPEC G ((!clipboard_has_first & isl1.status = met_criterion) -> F clipboard_has_first = TRUE);
LTLSPEC G ((!clipboard_has_first & isl2.status = met_criterion) -> F clipboard_has_first = TRUE);
LTLSPEC G ((!clipboard_has_first & isl3.status = met_criterion) -> F clipboard_has_first = TRUE);
LTLSPEC G ((!clipboard_has_first & isl4.status = met_criterion) -> F clipboard_has_first = TRUE);
LTLSPEC G ((clipboard_has_first & !clipboard_has_second & isl1.status = pause) -> F clipboard_has_second = TRUE);
```

```
LTLSPEC G ((clipboard_has_first & !clipboard_has_second & isl2.status = pause) -> F clipboard_has_second = TRUE);
LTLSPEC G ((clipboard_has_first & !clipboard_has_second & isl3.status = pause) -> F clipboard_has_second = TRUE);
LTLSPEC G ((clipboard_has_first & !clipboard_has_second & isl4.status = pause) -> F clipboard_has_second = TRUE);
--LTLSPEC G (isl1.status = met_criterion -> NF isl1.status = run | isl1.status = pause);
LTLSPEC G (isl1.status = exchange -> F isl1.status = met_criterion | isl1.status = pause);

MODULE island(clipboard_has_first, clipboard_has_second)
 VAR
  status                              :{run, pause, met_criterion, exchange} ;
 ASSIGN
  init(status) := run;
  next(status) :=
   case
    status = run : pause;
    status = pause & clipboard_has_first & !clipboard_has_second : exchange;
    status = pause & clipboard_has_second : {run, met_criterion};
    status = met_criterion & !clipboard_has_first : exchange;
    status = exchange & clipboard_has_first & clipboard_has_second : run;
    TRUE : status;
   esac;
 FAIRNESS
  Running
```

Проверим соответствие модели алгоритма спецификациям, с помощью программы NuSMV (рис. 4). Как видно из скриншота, алгоритм соответствует спецификациям.

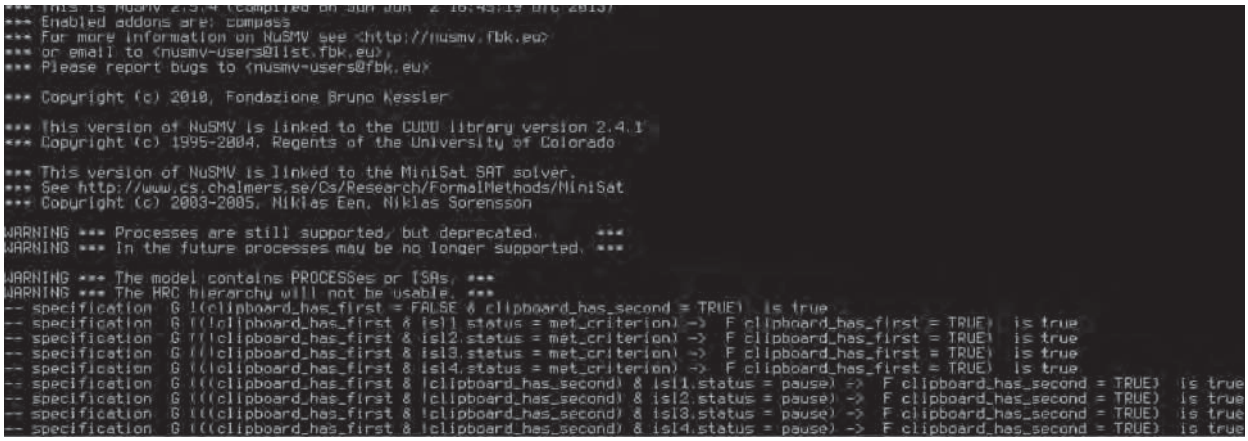

Рис. 4. Результат проверки в программе NuSMV

Заключение

Предложенный алгоритм может быть использован для составления расписания прохождения судов через отдельную шлюзованную систему. Он является разновидностью параллельных генетических алгоритмов и сочетает в себе преимущества островной и клеточной моделей. Среди преимуществ можно отметить возможность генерации приемлемых вариантов расписания уже с первой итерации. В алгоритме предусмотрена возможность значительного улучшения расписа-

ния благодаря добавлению дополнительных критериев оценки свободы и качества расположения групп судов в расписании.

Корректность данного алгоритма была проверена с помощью метода верификации на модели. Были составлены модель алгоритма и набор спецификаций, которым он дожен соответствовать. И с помощью системы NuSMV была произведена проверка предложенного алгоритма на корректность.

Список литературы

1. *Чебатуркин А. А.* Методы верификации конечных автоматов, взаимодействующих по акторной модели / А. А. Чебатуркин, М. А. Мазин. — СПб.: СПбИТМО, 2010.

2. NuSMV User Manual [Электронный ресурс]. — Электрон. дан. — Режим доступа: http://nusmv.irst.itc.it/NuSMV/userman/index-v2.html

3. Правила пропуска судов и составов через шлюзы внутренних водных путей Российской Федерации: утв. приказом Минтранса РФ от 24 июля 2002 г. № 100.

4. *Кононов В. В.* Гидротехнические сооружения водных путей, портов и континентального шельфа (Судоходный канал и бетонный шлюз) / В. В. Кононов. — СПб.: СПбГУВК, 2009.

УДК 621.396

А. А. Чертков,
канд. техн. наук, доцент
ГУМРФ имени адмирала С. О. Макарова;

Д. А. Загрединов,
аспирант,
ГУМРФ имени адмирала С. О. Макарова;

Ю. Б. Михайлов,
аспирант,
ГУМРФ имени адмирала С. О. Макарова

МОДЕЛЬ НЕЛИНЕЙНОЙ ЛОГИСТИЧЕСКОЙ СИСТЕМЫ АВТОМАТИЗАЦИИ ПЕРЕГРУЗОЧНОГО ПРОЦЕССА

MODEL OF NONLINEAR LOGISTICS SYSTEM OF AUTOMATION OF HANDLING PROCESS

В статье рассматривается модель и оптимизация решения нелинейной многомерной транспортной задачи, базирующейся на численных методах квадратичного программирования, реализуемого в вычислительной среде MATLAB. Достоверность результатов численного моделирования подтверждается экспериментом. По алгоритму произведен синтез логистической системы автоматизации перегрузочного процесса, обеспечивающей минимум транспортных расходов на управление ресурсами. Приведен пример расчета системы.