некотором положении ген маски принимает значение 1, то ген потомка в том же положении наследуется от первого родителя, если 0 – то второго.

Недостатки:

1. Проблема определения функции оценки пригодности. Решающее значение в эффективности генетических алгоритмов принимает функция оценки пригодности и схема кодирования. Общее правило построения функции оценки является то, что оно должно отражать пригодность хромосом посредством специализированных методов.
   К сожалению, пригодность хромосомы, рассчитанная специализированным методом, не всегда является полезной величиной для генетического поиска. В задачах комбинаторной оптимизации, где есть много ограничений, большинство точек в поисках места зачастую представляют недопустимые хромосомы и, следовательно, имеют нулевую пригодность.
2. Проблема локального оптимума заключается в том, что гены нескольких хромосом с сравнительно высокой пригодностью (но не оптимальной), особи могут доминировать в поколении, заставляя пригодность хромосом следующего поколения приблизиться к локальному максимуму. Как только популяция сошлась, способность генетического алгоритма найти лучшие решения фактически исключены, т.к. скрещивание почти идентичных хромосом производит мало нового. Остаются только мутации, которые просто выполняют медленный, случайный поиск. Для того, чтобы избежать таких ситуаций, мы должны изменить способ выбора особей для размножения.
3. Проблема завершения работы алгоритма. После многих поколений, популяция в большей степени сошлась, но глобальный максимум не достигнут. Средняя стоимость пригодности поколения будет высокой и разница пригодности между лучшими и средними особями будет небольшой. Следовательно, функция оценки пригодности следует оптимизировать, чтобы алгоритм достиг глобального максимума.

Генетические алгоритмы активно используются при оптимизации функций, при решении разнообразных задач на графах, задач компоновки и при настройке и обучении нейронных сетей.

Исследованиеподдержанопроектом CERES. Centers of Excellence for young RESearchers (Reg.no. 544137-TEMPUS-1-2013-SK-JPHES),

Список использованных источников:

1. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы — 2-е изд. — С. 452.
2. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. — 2-е изд. — С. 320.
3. Скобцов Ю. А. Основы эволюционных вычислений. — Донецк 2008. — С. 326.
4. Schmitt, Lothar M; Nehaniv, Chrystopher L; Fujii, Robert H (1998), Linear analysis of genetic algorithms, Theoretical Computer Science 208 – C. 111–148
5. Schmitt, Lothar M (2001), Theory of Genetic Algorithms, Theoretical Computer Science 259 – C. 1–61

АРХИТЕКТУРА REDUX

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Казаков С.Г.

Сиротко С.И. – канд. физ.-мат. наук, доцент

С развитием интернета, скорости соединения и возможностями браузеров, к одностраничным веб приложениям (SPA — single page application) увеличились требования, из-за чего нам необходимо управлять и поддерживать все больше и больше событий и состояний. Состояния включают в себя кэшировние данные, ответы сервера, а также данные созданные локально, но еще не сохраненные на сервере. Это касается и UI (User Interface) состояниям, таким как текущий путь (route), выбранная вкладка, отображение загрузки или навигация на страницах.

Основные проблемы современных приложения сводятся к увеличению сложности, что порождает ошибки программы из-за смешивания двух концепций асинхронность и изменение состояния. Другие решения справляются с проблемой на уровне отображения, убирая асинхронность и прямое манипулирование DOM. Архитектура Redux делает изменения состояния приложения предсказуемыми, путем введения некоторых ограничений на то, как и когда могут произойти обновления.

Redux является предсказуемым контейнером состояния для JavaScript приложений. Такая архитектура подразумевает создание нового состояния приложения при каждом изменении или действии, таким образом

Библиотека БГУИР

обеспечивается прозрачность, воспроизводимость и как следствие удобное тестирование. [1]

Используется три фундаментальных принципа для описания архитектуры:

1) Состояние всего вашего приложения сохранено в дереве объектов внутри одного хранилища.

2) Состояние только для чтения.

3) Все изменения состояния через чистые функции.

При использовании глобального состояния решается проблема общения компонентов приложения, вместо подписывания на события друг друга и все неявные связи используются действия. Действия порождают изменения состояния через чистые функции. Чистые функции это функции не зависят от окружающего состояния и среды выполнения, результат зависит только от входных параметров. Таким образом результат всегда предсказуемый – прозрачный.

Принцип коммуникации компонентов в архитектуре Redux представлен на рисунке 1:

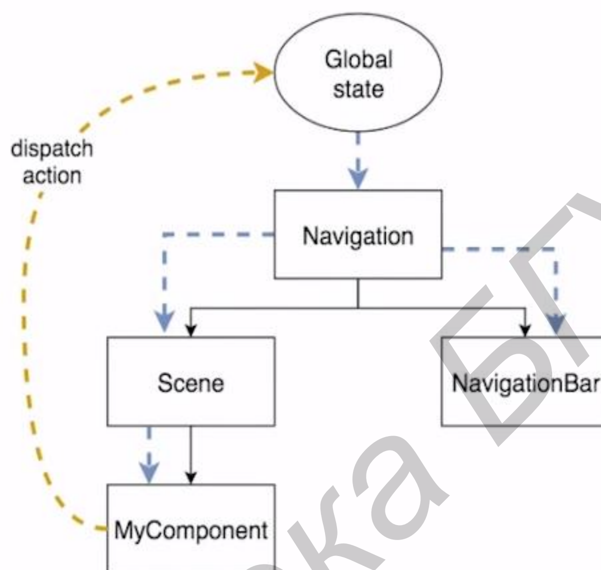

Рис. 1 – Схема общения между компонентами

Такой подход к созданию приложений дает возможность переиспользовать компоненты и делать их универсальными. Все состояние может быть сериализированона сервере и отправлено на клиент без особых усилий. Упрощается отладка приложения, когда мы имеем дело с одним состоянием, представленным деревом данных. Также можно сохранять состояние вашего приложения целиком для ускорения процесса разработки и использования подхода «горячей» перезагрузки. При использовании единственного дерева состояния, перемещение по истории вперед/назад получается без дополнительной реализации.

Основные понятия архитектуры [2]:

- Действия (Actions)
- Редюсеры (Reducers)
- Хранилище (Store)

Действия - это структуры, которые передают данные из вашего приложения в хранилище. Они являются единственными источниками информации для хранилища. Вы отправляете их в хранилище используя метод dispatch().

Действия описывают тот факт, что что-то совершилось, но не определяют, как в ответ изменяется состояние (state) приложения. Это работа для редюсеров (reducers).

Хранилище (Store) - это объект, который соединяет эти части вместе. Хранилище берет на себя следующие задачи:

1) Содержит состояние приложения (application state);

2) Предоставляет доступ к состоянию с помощью getState();

3) Предоставляет возможность обновления состояния с помощью dispatch(action);

4) Регистрирует слушатели (listeners) с помощью subscribe(listener).

Из материала выше очивидно, что использование архитектуры Reduxявляется залогом успешной разработки, отладки и тестирования. И поможет пересмотреть концепции и подходы к управлению данными.

Список использованных источников:

1. ReactRedux – MaxPatsitansky[Электронный ресурс], 2017. – Режим доступа: https://www.gitbook.com/book/maxfarseer/redux-course-ru/details. Дата доступа: 01.04.2017.
2. Redux – ConceptBook [Электронный ресурс], 2017. – Режим доступа: http://redux.js.org/index.html. Дата доступа: 02.04.2017.

Библиотека БГУИР