УДК 519.872.6

РАЗРАБОТКА АДАПТИВНОГО АЛГОРИТМА КОНТРОЛЕРА СВЕТОФОРА С ПРИОРИТЕТНОЙ ВЫБОРКОЙ В РЕЖИМЕ РЕАЛЬНОГО ВРЕМЕНИ

Х. Мирзаи

Воронежский государственный университет

Поступила в редакцию 07.12.2016 г.

Аннотация. В статье рассматривается задача авторегулирования светофора, на основе состояния транспортных потоков, полученных из некоторого детектора, в режиме реального времени. Характерной чертой предлагаемого алгоритма контроля светофора является адаптивность в реальном времени, а также масштабируемость при организации глобальной оптимизации множества светофоров, например для организации «зеленой волны». Особенностью этого алгоритма является способность подстраиваться под требующими обстоятельствами, когда нужно незамедлительно пропускать такие государственные службы, как пожарную и скорую помощь. Данный алгоритм протестирован в среде симуляции транспортных потоков SUMO (Simulation of Urban MObility) [1].
Ключевые слова: моделирование, транспортный поток, светофоры, авторегулирование, адаптивность в реальном времени, динамические фазы, SUMO.
Annotation. The article is devoted to the problem of auto-controlling the traffic lights, based on the traffic load status in real time received from a detector. The purposed traffic light controller algorithm is characterized as adaptive in real time, extendable for organizing global optimization with set of traffic lights, for instance, for organizing a green wave. Furthermore, the algorithm is capable to react to emergency situation like letting a fire truck or ambulance. It has been tested in SUMO (Simulation of Urban MObility).
Keywords: simulation, transport flow, traffic light, auto-controlling, adaptive in real time, dynamic phases, SUMO.

ВВЕДЕНИЕ

Затор является одной из основных причин опоздания и лишней траты топлива для водителей. Известно [2], что стратегия адаптивного регулирования светофора в реальном времени является самым большим потенциалом для устранения таких проблем. Однако, достижение масштабируемой оптимизации по всей сети остается сложной проблемой.

Для того чтобы достичь глобальной оптимизации во всей сети, сначала, следуя принципу «разделяй и властвуй», нужно достичь адаптируемого механизма управления светофором на уровне одного перекрестка. В данной статье наша цель – это разработка адаптивного алгоритма контролера светофора в режиме реального времени.

Основная проблема в традиционной системе управления светофорами заключается в том, что они работают в качестве закрытой системы по отношению к внешней среде. Другими словами, они не являются адаптивными к изменению состояния дороги. Такое управление можно назвать слепым, так как оно работает вне зависимости от настоящих внешних факторов, что приводит к неоптимальному использованию ресурса, т. е. в данном случае перекрестка. В результате, по некоторым направлениям появляется затор,

в то время как по другим направлениям возможна пустая очередь при зеленом сигнале. Адаптивные методы превращают статический светофор в живую систему, реагирующую на внешнюю среду.

Контролер от детекторов получает информацию о присутствии участников дорожного движения, чтобы настроить синхронизацию сигналов и фаз. Он может дать дополнительное время направлению, с высокой нагрузкой, а также уменьшить время или даже отменить фазу, в случае отсутствия трафика.

Детекторы могут быть сгруппированы в три класса: детекторы под проезжей частью, детекторы над проезжей частью, а также детекторы для обнаружения безмоторного движения [3].

Одной из главных проблем в организации адаптивных светофоров с помощью датчиков является их высокая стоимость и сложность установления. Одним из решений этой проблемы может стать видеодетекторы. Они снимают видео в режиме реального времени и передают поток снимков алгоритму компьютерного зрения для обнаружения количества автомобилей и состояний дороги. Стоимость установления такой системы более дешевая и простая в установлении. Существующие алгоритмы способны обнаруживать автомобилей в разных погодных условиях, даже в темные времена суток [4].

В данной статье, не обсуждается организация видеодетекторов. Предположим, что такая система существует и в реальном времени нам доступна информация о состоянии дорожного движения в любой момент.

Алгоритм работы светофора

С точки зрения программирования, подход к данной задаче можно назвать Агентно-ориентированным (Agent-oriented programming) [5]. Такой подход схож с реальной ситуацией, когда инспектор дорожного движения стоит в средине перекрестка и управляет движением транспортных средств из разных направлений. Под системой контроля светофора подразумевается «агент».

Задача агента заключается в оптимальном распределении права проезда транспортных потоков по перекрестку без конфликта. В данной задаче можно заметить два важных фактора: оптимальное распределение и отсутствие конфликта. Понятие оптимальности является абстрактным в данном случае, так как оно может определиться в зависимости от требований и ситуаций. С этой точки зрения появляются разные методы для реализации адаптивного алгоритма. В данной работе оптимальное распределение потоков включает в себя следующие принципы:

- предоставление дополнительного времени нуждающимся направлениям;
- установление максимального времени ожидания очереди;
- отключение зеленого сигнала для пустой очереди;
- избежание конфликтов;
- разрешение на пропуск максимально не конфликтующих потоков;
- гибкость и масштабируемость для глобальной оптимизации.

Модель перекрестка

Рассмотрим пример перекрестка с присоединяющимися к нему дорогами. Каждая дорога состоит из трех полос, как изображено на рис. 1.

Каждая стрелка показывает разрешенное направление движения, она называется связью между входящими и выходящими полосами с отношением одни ко многому. Обратите внимание на то, что обратное отношение не верно, так как в таком случае может произойти конфликтная ситуация между потоками.

Предположим, что каждой полосе назначен отдельный светофор с тремя сигналами (зеленый, красный и желтый). На практике один светофор может управлять несколькими направлениями движения.

В данной модели перекрестка каждая полоса имеет уникальный идентификатор. Связи проиндексированы, начиная с нуля, по направлению часовой стрелки. Таким образом, на рис. 1. индекс нуль присваивается первой полосе слева из входящей дороги на северном

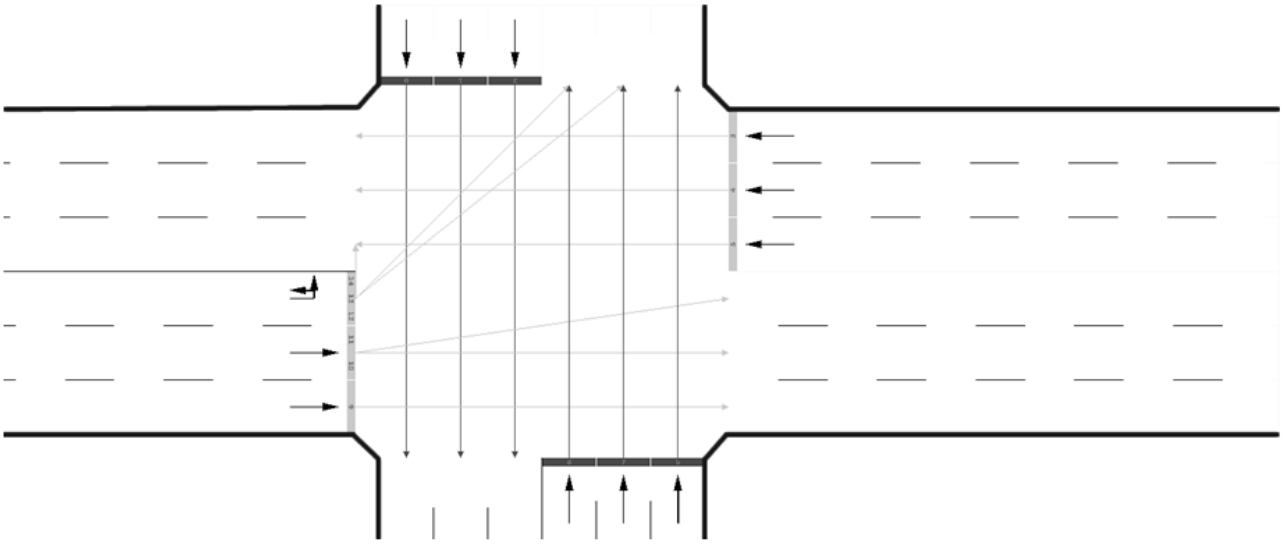

Рис. 1. Пример перекрестка. Стрелки показывают возможные направления транспортного потока

направлении перекрестка и заканчивается с полосой на западном направлении перекрестка с индексом 14, из имеющихся 15 связей.

Карта совместимости

Совместимость полос означает бесконфликтное движение между двумя полосами. В данной статье термин «дружба полос» может использоваться в качестве совместимости. Таким образом, определим некоторую структуру, содержащую информацию о совместимости полос. Карта совместимости – это некоторое множество из записей вида `C = {id: List<id>}`, где `id` – идентификатор полосы и `List<id>` – список из идентификаторов полос. Каждая запись в карте `C` определяет совместимость полосы с идентификатором `id` и другими полосами с идентификаторами `List<id>`, включая себя.

Динамическая группировка полос

Следующий блок кода показывает алгоритм динамической группировки полос. Метод `createGroup()` содержит логику составления группы из совместимых полос с заданным лидером. Под лидером подразумевается полоса с максимальной нагрузкой или длиной очереди.

Листинг 1. Метод создания группы

```
def createGroup(leaderId):
  group = []
  for id in IN_LANES_IDS:
    if areFriends(leaderId,id) and
isFriendOfGroup(id, group):
      group.append(id)
  return group
```

В листинге 1:

`leaderId` – идентификатор лидера.

`IN_LANES_IDS` – список идентификаторов полос с входящими потоками.

`areFriends(id,id)` – проверяет дружбу между двумя полосами.

`isFriendOfGroup(id,group)` – проверяет дружбу между заданной полосой с группой.

Листинг 2. Поиск лидера

```
def findLeader(queues):
  leader = IN_LANES_IDS[0]
  for id in set(IN_LANES_IDS):
   if rank(id,queues) >
              rank(leader,queues):
     leader = id
  return leader
```

Листинг 3. Определение метрика (ранка)

```
def rank(id,queues):
  return priority[id]*queues[id]
```

В листинге 3:

`queues` – структура, содержащая текущие длины очередей,

`proiority` – структура, содержащая приоритет для каждой полосы.

Функция `rank(id)` играет важную роль при поиске лидера. Данная функция является абстрактной, в том смысле, что реализация зависит от контекста и требования системы. Эта функция возвращает некоторую оценку, с которой можно сравнить полосы друг с другом. Например, метрикой может стать просто длина очереди, умноженная на некоторый коэффициент, означающий приоритет и т. п.

При поиске лидера часто возникает такой случай, когда некоторые полосы имеют одинаковые длины. Значение приоритета помогает уточнить такую ситуацию. Стоит отметить, что операция умножения в функции 3. выбрана не случайно. Когда очередь с идентификатором `id` пустая, очевидно, что ее оценка должна равняться нулю вне зависимости от значения приоритета. Таким образом, мы исключаем пустые очереди при поиске лидера.

Кроме того, работа с приоритетом может принести гибкость в алгоритме. В начале статьи говорилось о том, что система должна быть гибкой для работы с глобальным оптимизатором, например, в случае, когда необходимо организовать «зеленую волну» (последовательность зеленых сигналов иногда называют «зеленым коридором»). Другим немаловажным случаем является необходимость пропустить пожарную или скорую помощь. Таким образом, достаточно повысить приоритет до нужного значения для требующей полосы, вследствие чего система автоматически адаптируется.

Таким образом, необходимо иметь подсистему обновления приоритетов после перегруппировки.

Листинг 4. Обновление приоритетов

```
def updatePriorities(group):
  for id in IN_LANES_IDS:
    if id in group: priority[id] = 1
    else:
     if (priority[id]+1)*MIN_GREEN_
TIME > MAX_RED_TIME:
      priority[id] += HIGH_PROIORITY
     else:
      priority[id] += 1
```

В листинге 4:

`HIGH_PROIORITY` – некоторое значение больше 1, например 100.

Функция `updatePriorities(group)` демонстрирует реализацию подсистемы обновления приоритетов. Логика данной функции заключается в следующем.

Всем полосам, входящие в текущую группу с зеленым сигналом, присваивается приоритет со значением 1, а остальным увеличивается на единицу. Если суммарное время ожидания за красным сигналом будет превышать значение порога `MAX_RED_TIME`, то значение приоритета увеличивается на значение `HIGH_PROIORITY`. Это гарантирует, что время красного сигнала не будет не дольше значения порога `MAX_RED_TIME`. При каждой итерации, значение приоритета полосы, не входящей в текущую группу, увеличивается, чтобы получить зеленый сигнал при последующей итерации. Такая логика позволяет пропустить (не давать им зеленый сигнал) полосы с пустыми очередями, и дать больше время полосам с высшим приоритетом.

Обновление программы светофора

В отличие от традиционного светофора с фиксированными фазами, в данной работе фазы определяются несколько иначе. Обычно светофор имеет более двух фаз, например, перекресток со схемой, изображенной на рис. 1, должна иметь минимум четыре фазы: две основных и две промежуточных. В данной работе имеется две фазы: одна основная, а другая промежуточная для любых схем перекрестка. При основной фазе включается зеленый сигнал полосам, которые входят в группу. Промежуточная фаза предназначена для безопасной смены одной группы на другую.

Листинг 5. Смена одной группы на другую.

```
def updateProgram():
  if existsAtLeastOneCar():
    program = getProgram("current_
program")
    leaderId = findLeader(queues)
    newGroup = createGroup(leaderId)
    updatePriorities(newGroup)
    updatePhases(program,newGroup)
```

Функция `updateProgram()` показывает логику обновления программы контролера светофора. Функция `existsAtLeastOnCar()` возвращает булево значение о том, существует ли хотя бы один автомобиль. Такая проверка предназначена для избегания от ненужного обновления программы контролера. Такая ситуация возникает тогда, когда автомобиль не попадает в поле зрения видеодетектора. В таком случае лучше придерживаться текущей программы. Функция `updatePhases()` возвращает две фазы, о которых шла речь.

Реализация в среде SUMO

Немецкий центр авиации и космонавтики (DLR) – национальный центр аэрокосмических, энергетических и транспортных исследований Германии. SUMO является программным комплексом и продуктом института транспортных систем в DLR с открытым исходным кодом, который доступен с 2001 года.

«Моделирование городского транспорта», или SUMO (Simulation of Urban MObility) является микроскопическим, мультимодальным моделированием трафика. Моделирование в SUMO является чисто микроскопическим: каждое транспортное средство моделируется в явном виде, имеет собственный маршрут, и перемещается по отдельности через сеть.

Несмотря на хорошую скорость работы симулятора и предоставляющие инструменты, к сожалению, SUMO мало документировано. Для проведения эксперимента в данной среде использовалась только официальная документация, которая доступна из [1].

Тем не менее, SUMO является мощной средой для моделирования городских транспортных потоков. В данный момент не существует других бесплатных симуляторов с открытым исходным кодом, способных конкурировать с SUMO. В данной статье мы не будем обсуждать правила использования SUMO, так как она требует отдельной статьи.

Результаты сравнения в среде SUMO

Сравнение двух методов сделано в середе SUMO. Схема перекрестка приведена на рис. 1. Все дороги состоят из трех полос.

В среде SUMO можно определить транспортный поток по одному автомобилю или сгенерировать поток случайным образом. Случайные потоки генерируются с заданной вероятностью по биномиальному закону распределения, который приближается к пуассоновскому закону распределения при малых значениях вероятности. Генерация потоков для обоих методов выполнена одинаково. В табл. 1 приведены результаты тестирования.

В данном эксперименте одним из важных выходных параметров является WaitingTime – показывает среднее время проезда автомобиля со скоростью меньше 0.1 м/с, другим не менее значимым параметром является TimeLoss – время, когда водитель вынужден ехать меньше идеальной скорости.

Статическая программа состоит из шести фаз: три основных с длиной 33 секунд и три промежуточных с длиной 6 секунд. Далее приведено определение программы контролера со статическими фазами.

Листинг 6. Определение программы контролера

```
<tlLogic id="0" type="static"
offset="0">
 <phase duration="33"
   state="GGGrrrGGGrrrrrr"/>
 <phase duration="6"
   state="yyyrrryyyrrrrrr"/>
 <phase duration="33"
   state="rrrGGGrrrGGGrrr"/>
 <phase duration="6"
   state="rrryyyrrryyyrrr"/>
 <phase duration="33"
   state="rrrrrrrrrGGGGGG"/>
 <phase duration="6"
   state="rrrrrrrrryyyyyy"/>
</tlLogic>
```

Каждый символ в атрибуте `state` означает один из трех сигналов светофора, зеленый (G), красный (r) и желтый (y). Позиция символов соответствует индексу связи между входящими и выходящими полосами. Например, первая фаза с `state="GGGrrrGGGrrrrrr"`

Таблица 1

Выходные данные симулятора с адаптивной программой против статической программы контролера светофора

Адаптивная программа	Статическая программа
Simulation started with time: 0.00 Simulation ended at time: 4113.00 Reason: TraCI requested termination. Performance: Duration: 68631ms Real time factor: 59.9292 UPS: 3251.300433 Vehicles: Inserted: 1943 Running: 0 Waiting: 0 Statistics (avg): RouteLength: 981.85 Duration: 114.84 WaitingTime: 45.04 TimeLoss: 94.51 DepartDelay: 1172.12	Simulation started with time: 0.00 Simulation ended at time: 9472.00 Reason: TraCI requested termination. Performance: Duration: 17878ms Real time factor: 529.813 UPS: 65762.109856 Vehicles: Inserted: 1952 Running: 0 Waiting: 0 Teleports: 12 (Wrong Lane: 12) Emergency Stops: 1 Statistics (avg): RouteLength: 981.74 Duration: 602.30 WaitingTime: 454.46 TimeLoss: 581.77 DepartDelay: 2532.98

означает включение зеленого сигнала на 33 секунды для связи с индексом 0-2 и 6-8 и красного сигнала для остальных. Потом идет промежуточная фаза на 6 секунд для связей с зеленым сигналом в текущей фазе и т. д.

ЗАКЛЮЧЕНИЕ

В данной работе представлен адаптивный метод авторегулирования светофоров в реальном времени. Разработанный метод протестирован в среде симулятора транспортных потоков SUMO. Результаты тестирования показывают значительное преимущество и эффективность работы представленного адаптивного метода.

Из преимуществ данного метода можно выделить быстродействие в режиме реального времени, а также простоту и оптимальную стоимость организации в реальной ситуации. Кроме того, благодаря механизму назначения приоритета, позволяющему легко организовывать зеленую волну, такой подход является гибким для глобальной оптимизации.

Несмотря на хорошие результаты, полученные при тестировании в среде SUMO, необходимо более детально тестировать и развивать метод с учетом настоящих факторов и ситуаций на дорогах. В дальнейшем планируется использовать модель для глобальной оптимизации светофоров по всей сети.

СИПИСОК ЛИТЕРАТУРЫ

1. Официальная документация для пользователя. – URL: http://www.sumo.dlr.de/userdoc/ (дата обращения: 28.09.2016)

2. *Xiao-Feng Xie, Stephen F.* Schedule-Driven Coordination for Real-Time Traffic Network Control [Электронный ресурс]. 2012. The Robotics Institute, Carnegie Mellon University. – URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4701/4744.pdf (дата обращения: 28.09.2016)

3. Traffic light control and coordination. – URL: https://en.wikipedia.org/wiki/Traffic_light_control_and_coordination#Dynamic_control (дата обращения: 28.09.2016)

4. *Kostia Robert*. Video-based traffic monitoring at day and night time Vehicle Features Detection and Tracking. Smart Transport and Road – Sensors and Surveillance. NICTA (National ICT Australia). – URL: http://www.nicta.com.au/pub?doc=2128 (дата обращения: 28.09.2016)

5. Агентно-ориентированный подход. – URL: https://en.wikipedia.org/wiki/Agent-oriented_programming (дата обращения: 28.09.2016)

Хассан Мирзаи – аспирант, кафедра программирования и информационных технологий, факультет компьютерных наук, Воронежский государственный университет.
E-mail: hasanmirzae@yahoo.com

Hassan Mirzaee – Ph.D. student, Department of Programming and Information Technologies, Faculty of Computer Science, Voronezh State University.
E-mail: hasanmirzae@yahoo.com