**Алгоритм LKH**

Алгоритм 2-opt является частным случаем алгоритма*-opt*, где на каждом шаге λ звеньев текущего тура заменяются на λ звеньев таким образом, чтобы достигался более короткий тур. Другими словами, на каждом шаге тур становится более коротким путем удаления λ звеньев и объединения полученных путей в новый, возможно, изменив один или несколько из них.

λ - opt алгоритм основан на концепции*-оптимальности*:

**Тур считается *-оптимальным* (или просто *-оптимальным* ), если невозможно получить более короткий тур, заменив любое  его λ звеньев любым другим набором λ звеньев.**

Из этого определения очевидно, что любой λ - оптимальный тур также является λ'-оптимальным для 1 ≤ λ’ ≤ λ. Также легко видеть, что тур, содержащий n городов, оптимален, если и только если это n-оптимально.

В общем, чем больше значение λ, тем больше вероятность того, что последний тур

оптимален. При достаточно больших λ кажется, по крайней мере интуитивно, что λ - оптимальный тур должен быть оптимальным.

К сожалению, количество операций по проверке всех λ - обменов быстро увеличивается

по мере увеличения количества городов. В наивной реализации тестирование λ -обмен имеет временную сложность O (n^λ). Кроме того, нет нетривиальной оценка сверху количества λ - обменов. В результате значения λ = 2 и λ = 3 являются наиболее часто используемыми. В одном исследовании были использованы значения λ = 4 и λ = 5 [16].

Однако недостатком является то, что λ нужно указывать заранее. Это трудно

узнать, какой λ нужно использовать для достижения наилучшего компромисса между временем работы и качеством решения.

Лин и Керниган устранили этот недостаток, введя мощную *переменную λ -opt* алгоритм. Алгоритм изменяет значение λ во время своего выполнения,

решая на каждой итерации, каким должно быть значение λ. На каждой итерации

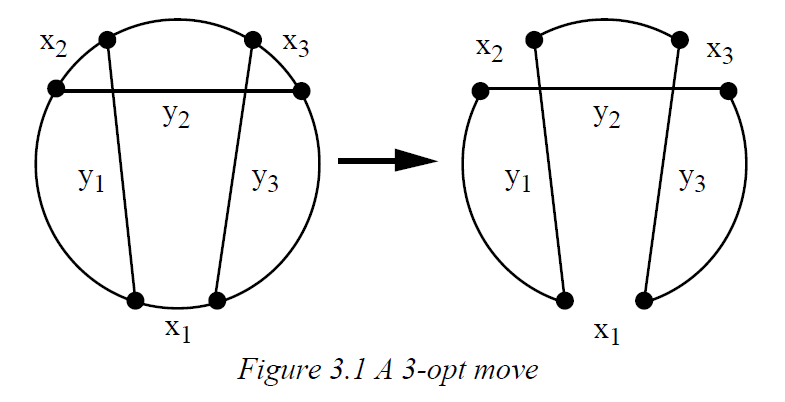
алгоритм проверяет, для возрастающих значений λ, приведет ли перестановка

λ звеньев к более короткому туру. Учитывая, что рассматривается обмен r звеньями, проводится серия тестов, чтобы определить, следует ли рассматривать обмен r+1 звеньями. Это продолжается пока не будут выполнены некоторые условия остановки.

На каждом шаге алгоритм учитывает растущий набор потенциальных обменов (начиная с r = 2). Эти обмены выбраны таким образом, чтобы возможный тур мог быть сформирован на любом этапе процесса. Если в ходе исследования удастся найти новый более короткий тур, то фактический тур заменяется новым туром.

Алгоритм Лина-Кернигана относится к классу так называемых алгоритмов *локальной  
оптимизации* [17, 18]. Алгоритм указан в терминах *обмена* (или *ходы*), которые могут преобразовать один тур в другой. Учитывая возможный тур, алгоритм неоднократно выполняет обмены, которые сокращают продолжительность текущего тура, пока не будет достигнут тур, для которого обмен не приведет к улучшению. Этот процесс может повторяться много раз из начальных туров, сгенерированных каким -либо рандомизированным способом. Алгоритм более подробно описан ниже.

Пусть T будет текущим туром. На каждой итерации алгоритм пытается найти  
два набора звеньев, X = {x\_1, … , x\_r} и Y = {y\_1, … , y\_r}, таким образом, что, если  
звенья X будут удалены из T и заменены звеньями Y, то результатом будет  
лучший тур. Этот обмен ссылками называется *движением r-opt.* Рисунок 3.1  
иллюстрирует движение 3-opt.

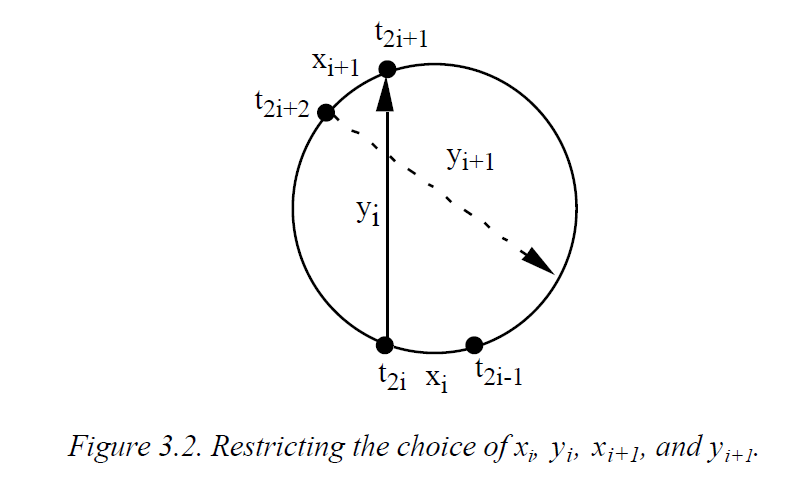


Два набора X и Y строятся поэлементно. Изначально X и Y пусты. На шаге iпара звеньев, x\_i и у\_i, добавляются в X и Y соответственно.

Для достижения достаточно эффективного алгоритма в X и Y могут входить только звенья, соответствующие следующим критериям:

*(1) Критерий последовательного обмена*

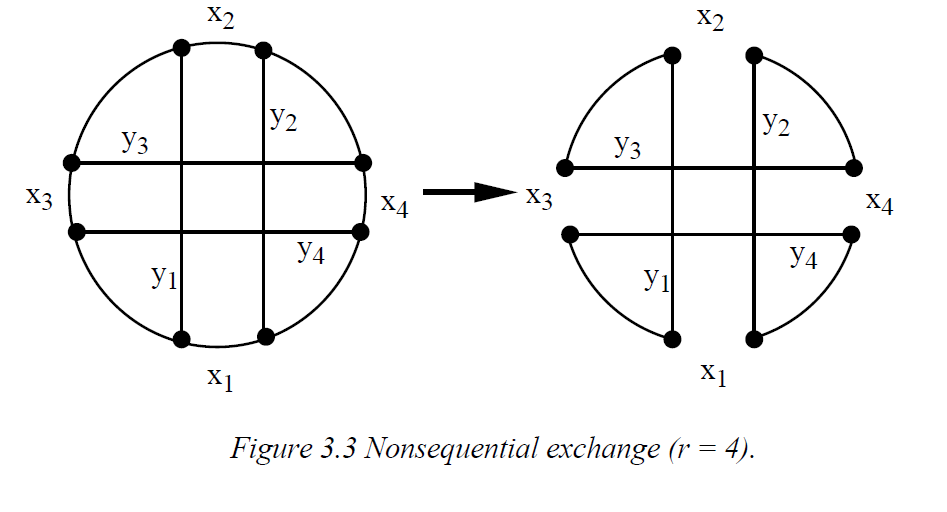
x\_i и у\_i должны иметь общую конечную точку, как и y\_i и x\_(i+1). Если t\_1 обозначает один из двух концов x\_1, то в общем случае имеем: x\_i = (t\_(2i-1), t\_2i), y\_i = (t\_2i, t\_(2i+1)) и x\_(i+1) = (t\_(2i+1), t\_(2i+2)) для i ≥ 1. См. Рисунок 3.2.



Как видно, последовательность (x\_1, y\_1, x\_2, y\_2,…, x\_r, y\_r,) представляет собой цепочку смежных звеньев.

Необходимым (но не достаточным) условием того, что обмен звеньями X с звеньями Y приведет к туру, в котором цепочка замкнута, т. е. y\_r = (t\_2r, t\_1). Такой обмен называется последовательным (***sequential***).

Как правило, улучшение тура может быть достигнуто путем последовательного  
обмена соответствующей нумерацией затронутых звеньев. Однако это не  
всегда так. На рис. 3.3 показан пример, в котором последовательный обмен  
невозможен.



*(2) Критерий осуществимости*

Требуется, чтобы x\_i = (t\_(2i-1), t\_2i) всегда выбирался так, чтобы, если t\_2i соединяется с t\_1, то итоговая конфигурация - это тур. Этот критерий осуществимости используется для i ≥ 3 и гарантирует, что можно ***вплотную приблизиться***ктуру. Этот критерий был включен в алгоритм как для сокращения времени выполнения, так и для упрощения реализации.

*(3) Критерий положительного усиления*

Требуется, чтобы y\_i всегда выбирался так, чтобы *коэффициент усиления(gain)*, G\_i, из предложенного набора обменов являлся положительным. Предположим, что g\_i = c(x\_i) - c(y\_i) является улучшением при замене x\_i на y\_i. Тогда G\_i равна сумме g\_1 + g\_2 + … + g\_i.

Этот критерий остановки играет большую роль в эффективности алгоритма.  
Требование, чтобы каждая частичная сумма, G\_i, была положительной, сразу кажется, что это слишком ограничительно. Однако то, что это не так, следует из следующего простого факта:

***Если последовательность чисел имеет положительную сумму, существует циклическая перестановка этих чисел, так что каждая частичная сумма положительна***.

Доказательство простое и может быть найдено в [1].

*(4) Критерий дизъюнктивности*

Наконец, требуется, чтобы множества X и Y не пересекались. Это упрощает реализацию, сокращает время выполнения и обеспечивает эффективный критерий остановки.

Ниже приводится краткое описание **основного алгоритма** (*упрощенная версия исходного алгоритма*).

1. Сгенерируем случайный начальный тур T.

2. Пусть i = 1. Выберем t1

3. Выберем x1 = (t\_1, t\_2) ∈ T

4. Выберем y1 = (t\_2, t\_3) ∉ 𝑇 так, чтобы G1> 0. Если это невозможно, переходим на **шаг 12**

5. Пусть i = i + 1

6. Выберем x\_i = (t\_(2i-1), t\_2i) ∈ T так, чтобы

(a) Если t\_2i присоединен к t\_1, тогда исходом в результате будет тур T` и

(b) x\_i ≠ y\_s для всех s < i

Если тур T` лучше тура T, то T = T`, и мы переходим **на шаг 2**

7. Выберем 𝑦\_𝑖 = (𝑡\_2𝑖, 𝑡\_(2𝑖+1)) ∉ 𝑇 так, чтобы

(a) 𝐺\_𝑖 > 0

(b) 𝑦\_𝑖 ≠ 𝑥\_s для всех s ≤ i, и

(c) x\_(i+1) существует

Если такой y\_i существует, то переходим на **шаг 5**

8. Если есть неиспытанный вариант для y\_2, пусть i = 2, и переходим к **шагу 7**

9. Если есть неиспытанный вариант для x\_2, пусть i = 2, и переходим к **шагу 6**

10. Если есть неиспытанный вариант для y\_1, пусть i = 1, и переходим к **шагу 4**

11. Если есть неиспытанный вариант для x\_1, пусть i = 1, и переходим к **шагу 3**

12. Если есть неиспытанный вариант для t\_1, то переходим к **шагу 2**

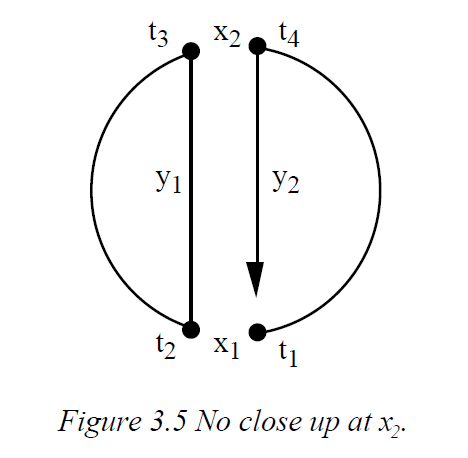
13. Стоп (или переходим на **шаг 1**)

*Комментарии к алгоритму:*

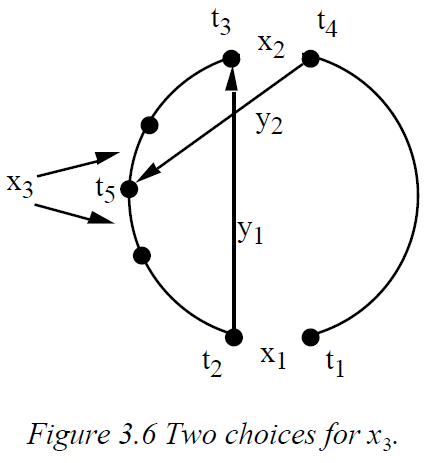
**Шаг 1.** В качестве отправной точки для исследований выбирается случайный тур.

**Шаг 3.** Выберите звено x\_1 = (t\_1, t\_2) в туре. Когда выбран t\_1, есть два варианта для x\_1. Здесь глагол “выбрать” означает “*выбрать неиспытанный вариант*”. Однако каждый раз, когда было найдено улучшение тура (на **шаге 6**), все альтернативы считаются неиспробованными.

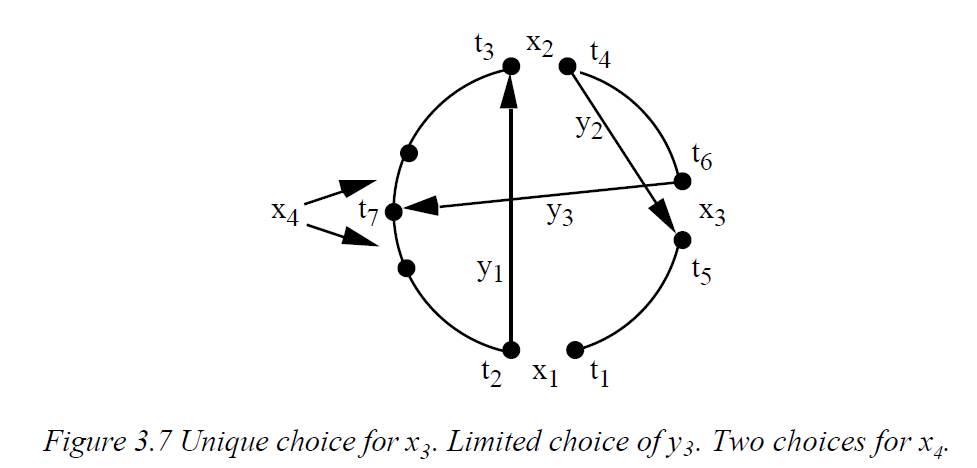
**Шаг 6.** Есть два варианта x\_i. Однако для данного y\_(i-1) (i ≥ 2) только один из них позволяет “закрыть” тур (путем добавления y\_i). Другой выбор приводит к двум несвязанным подтурам. Однако только в одном случае допускается такой неосуществимый выбор, а именно при i = 2. На рисунке показана эта ситуация.



Если y\_2 выбран так, чтобы t\_5 лежал между t\_2 и t\_3, то тур может быть закрыт на следующем шаге. Но тогда t\_6 может находиться по обе стороны от t\_5 (см. Рисунок 3.6); оригинальный алгоритм исследовал обе альтернативы.



С другой стороны, если y2 выбран так, чтобы t5 лежал между t4 и t2, для t6 есть только один выбор (он должен лежать между t4 и t5), а t7 должен лежать между t2 и t3. Но тогда t8 может находиться по обе стороны от t7. Оригинальный алгоритм исследовал альтернативу, для которой c(t7, t8) является максимальным.



Условие (b) на шаге 6 и шаге 7 гарантирует, что множества X и Y не пересекаются: y\_i не должен быть ранее разорванной связью, а x\_i не должен быть ранее добавленной связью.

Шаги 8-12. Эти шаги вызывают откат назад. Обратите внимание, что возврат разрешен только в том случае, если улучшения не обнаружено, и только на уровнях 1 и 2.

Шаг 13. Алгоритм завершается туром решения, когда все значения t\_1 были проверены без улучшения. При необходимости на шаге 1 может быть рассмотрен новый случайный начальный тур.

Описанный выше алгоритм отличается от оригинального своей реакцией на улучшения тура. В приведенном выше алгоритме тур T заменяется более коротким туром T’, как только будет найдено улучшение (на шаге 6). В отличие от этого оригинальный алгоритм продолжает свои шаги, добавляя потенциальные обмены, чтобы найти еще более короткий тур. Когда больше обмены невозможны или когда Gi ≤ G\*, где G\* - лучшее улучшение T, зарегистрированное на данный момент, поиск прекращается, и текущий тур T заменяется наиболее выгодным туром. В своей статье [1] Лин и Керниган не изложили причины внедрения этого метода. Это усложняет реализацию и не приводит ни к лучшим решениям, ни к сокращению времени выполнения.

*Уточнения Лина и Кернигана*

Узким местом алгоритма является поиск звеньев для входа в множества X и Y. Поэтому для повышения эффективности следует проявлять особую осторожность, чтобы ограничить этот поиск. Следует рассматривать только те обмены, которые имеют разумные шансы привести к сокращению продолжительности тура.

Базовый алгоритм, представленный в предыдущем разделе, ограничивает поиск с  
помощью следующих четырех правил:

(1) Допускаются только последовательные обмены.

(2) Предварительный выигрыш должна быть положительной.

(3) Тур может быть "закрытым" (за одним исключением, i = 2).

(4) Нельзя добавлять ранее неработающее звено, и ранее добавленное звено не должно быть разорвано.

Чтобы еще больше ограничить поиск, Лин и Керниган усовершенствовали алгоритм, введя следующие правила:

(5) Поиск звена для входа в тур, y\_i = (t\_2i, t\_(2i+1)), ограничивается пятью ближайшими соседями t\_2i.

(6) Для i ≥ 4 ни одно звено x\_i в туре не должно быть разорвано, если это обычное звено на небольшое количество (2-5) туров по решению.

(7) Поиск улучшений прекращается, если текущий тур является тем же, что и в предыдущем туре по решениям.

Правила 5 и 6 являются эвристическими правилами. Они основаны на ожиданиях того, какие звенья, скорее всего, будут относиться к оптимальному туру. Они экономят рабочее время, но иногда за счет того, что не достигают наилучших возможных решений.

Правило 7 также экономит время выполнения, но не влияет на качество найденных решений. Если тур совпадает с предыдущим туром по решению, нет смысла пытаться улучшить его дальше. Таким образом, время, необходимое для проверки того, что дальнейшие улучшения невозможны (время оформления заказа), может быть сэкономлено. По словам Лина и Кернигана, сэкономленное таким образом время обычно составляет от 30 до 50 процентов рабочего времени.

В дополнение к этим уточнениям, целью которых в первую очередь является *ограничение* поиска, Лин и Керниган добавили некоторые уточнения, целью которых в первую очередь является *направление* поиска. Там, где у алгоритма есть выбор альтернатив, эвристические правила используются для определения приоритетов этих альтернатив. В тех случаях, когда необходимо выбрать только одну из альтернатив, наиболее приоритетной является

выбранный. В случаях, когда необходимо попробовать несколько альтернатив, альтернативы рассматриваются в порядке убывания приоритета (с использованием обратного отслеживания). Чтобы быть более конкретным, используются следующие правила:

(8) Когда должно быть выбрано звено y\_i (i≥2), каждому возможному выбору присваивается приоритет c(x\_(i+1)) – c(y\_i).

(9) Если есть две альтернативы для x\_4, то выбирается то, где c(x\_4) является наибольшей.

Правило 8 - это эвристическое правило для ранжирования звеньев, которые будут добавлены в Y. Приоритетом для y\_i является длина следующего (уникального) разорванного звена, x\_(i+1), если y\_i включён в тур, то вычитаем длину y\_i. Таким образом, алгоритм получает некоторое представление о будущем. Путем максимизации величины c(x\_(i+1)) – c(y\_i), алгоритм стремится разорвать длинное звено и включить короткое звено.

Правило 9 касается особой ситуации на рисунке 3.7, в которой есть два варианта x\_4. В этом случае правило отдает предпочтение самому длинному звену. В трёx других случаях, а именно для x\_1, x\_2, и иногда и x\_3 (см. рис 3.6) существуют  
две доступные альтернативы. В этих ситуациях алгоритм рассматривает оба  
варианта с использованием обратного отслеживания (если только не был найден улучшенный тур). В своей статье Лин и Керниган не указывают последовательность, в которой рассматриваются альтернативы.

В качестве последнего уточнения Лин и Керниган включили ограниченную защиту от ситуаций, когда только непоследовательные обмены могут привести к лучшему решению. После того, как локальный оптимум найден, алгоритм проверяет среди  
разрешенных к разрыву звеньев, возможно ли дальнейшее улучшение  
путем несогласованного изменения 4 опций (как показано на рисунке 3.3). Лин и  
Керниган отметили, что эффект этой процедуры последующей оптимизации  
существенно варьируется от проблемы к проблеме. Однако время, затраченное на тест, невелико по сравнению с общим временем выполнения, поэтому это дешевая страховка.

**Модифицированный алгоритм Лина-Кернигана**

Оригинальный алгоритм Лина и Кернигана был достаточно эффективным. Для  
задач с числом городов до 50, вероятность получения оптимальных решений в  
одном испытании была близка к 100 процентам. Для проблем со 100 городами  
вероятность снизилась до 20-30 процентов. Однако, проведя несколько испытаний, каждый раз начиная с нового случайного тура, можно было найти оптимальное решение этих проблем с почти 100-процентной уверенностью.

Алгоритм был оценен по целому спектру задач, среди которых задача бурения с 318 точками. Из-за ограничений компьютерной памяти проблема была разделена на три более мелкие проблемы. Тур по решению был получен путем раздельного решения подзадач и, наконец, объединения трех туров. В то время, когда Лин и Керниган написали свою статью (1971), оптимальное решение этой  
проблемы было неизвестно. Теперь, когда оптимум известен, можно отметить, что  
их решение было на 1,3 процента выше оптимального.

Ниже представлена модифицированная и расширенная версия их алгоритма. Новый алгоритм является значительным улучшением исходного  
алгоритма. Например, для упомянутой проблемы 318 городов оптимальное решение теперь найдено в нескольких испытаниях (примерно 2) и за очень короткое время (около одной секунды на 300 МГц G3 Power Macintosh). В целом, качество решений, достигаемых алгоритмом, очень впечатляет. Алгоритму удалось найти оптимальные решения для всех примеров проблем, которые мы  
смогли получить, включая проблему 7397 городов (самый большой нетривиальный экземпляр проблемы, решенный до оптимальности сегодня).

Повышение эффективности в первую очередь достигается за счет пересмотра эвристических правил Лина и Кернигана для ограничения и направления поиска. Даже если их эвристические правила кажутся естественными, критический анализ показывает, что они страдают значительными недостатками.

*4.1 Наборы кандидатов*

Центральным правилом в исходном алгоритме является эвристическое правило, которое ограничивает  
включение ссылок в тур пятью ближайшими соседями по данному городу (Правило 5  
в разделе 3.2). Это правило ориентирует поиск на короткие туры и существенно сокращает  
усилия по поиску. Однако существует определенный риск того, что  
применение этого правила может помешать поиску оптимального решения. Если  
оптимальное решение содержит одно звено, которое не связано с пятью ближайшими  
соседями двух его конечных городов, то алгоритм столкнется с трудностями в  
получении оптимального.

Неадекватность этого правила особенно ярко проявляется в крупных  
проблемах. Например, для задачи с 532 городами [19] одним из звеньев в оптимальном  
решении является 22-й ближайший соседний город для одной из его конечных точек. Поэтому для  
того, чтобы найти оптимальное решение этой проблемы, число ближайших-