用户均衡条件（固定需求情形）

$$\left(c_{r,w}-\mu_w\right)f_{r,w}=0,\ \ r\in R_w, w\in W$$
$$c_{r,w}-\mu_w\geq 0,\quad f_{r,w}\geq 0,\ \ r\in R_w, w\in W$$
$$\sum_{r\in R_w} f_{r,w}=d_w, w\in W$$

$$x_a=\sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r}, a\in A$$
$$c_{r,w}=\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}, r\in R_w, w\in W$$

$$\left(\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}-\mu_w\right)f_{r,w}=0,\ \ r\in R_w, w\in W$$
$$\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}-\mu_w\geq 0,\ \ f_{r,w}\geq 0,\quad r\in R_w, w\in W$$
$$\sum_{r\in R_w} f_{r,w}=d_w,\ \ w\in W$$
$$x_a=\sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\quad a\in A$$

用户均衡下的交通流分配算法

- ➢All-or-Nothing traffic assignment
- ➢Capacity constraint traffic assignment
- ➢Incremental assignment

} 启发式

- ➢Beckmann’s tranformation

All-or-Nothing Traffic Assignment

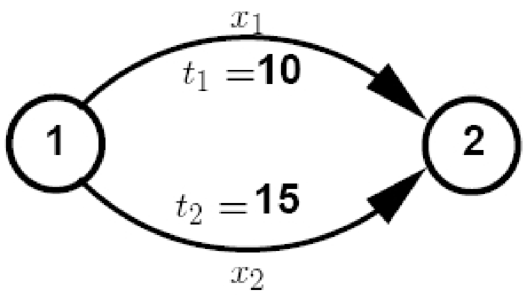
- “All-or-Nothing” is a commonly used procedure for network loading. This does not recognize the dependence between flows and travel time (travel time is assumed to be fixed).

- In this method the trips from any origin zone to any destination zone are loaded onto a single, minimum cost, path between them.

- It involve the technique of network loading- the process of assigning the O-D entries to the network for specific link travel times by following the route choice criterion.

Example

To demonstrate how this assignment works, an example network is considered. This network has two nodes having two paths as links. Let us suppose a case where travel time is not a function of flow as shown in other words it is constant as shown in the figure below.



- This model is unrealistic as only one path between every O-D pair is utilized even if there is another path with the same or nearly same travel cost;
- Traffic on links is assigned without consideration of whether or not there is adequate capacity or heavy congestion;
- Travel time is a fixed input and does not vary depending on the congestion on a link.

Capacity Constraint Traffic Assignment

- As traffic volume increases, speed decreases.
- BPR (Bureau of Public Roads) function

$$t_a = t_0\left(1 + 0.15\left(\frac{x_a}{C_a}\right)^4\right)$$

t_a	travel time at traffic flow x_a
t_0	zero-flow or free flow travel time
x_a	traffic flow on link *a* (veh/hr)
C_a	practical link capacity (veh/hr)
α, β	parameters

Capacity Constraint Traffic Assignment

- Incorporating capacity restraint. The most common method is to load the network and adjust assume link speeds after each loading to reflect volume/ capacity restraints.
- These loading and adjustments are introduced until a balance is obtained.
- Experience shows that a balance can be normally achieved after 3-4 loadings.
- This process attempt to capture the equilibrium nature of the traffic assignment problem. It is a more realistic representation of traffic and is in widespread use.

Capacity Constraint Traffic Assignment

Algorithm

Step 0 *Initialization*. Perform All-or-Nothing based on $t_a^0 = t_a(0), \forall a \in A$, and obtain link flow $\{x_a^0\}$. Set iteration counter $n = 1$.

Step 1 *Updating*. Set $t_a^0 = t_a\left(x_a^{n-1}\right), \forall a \in A$.

Step 2 *Network loading*. Assign all trips to the network using all-or-nothing based on the travel time $\{t_a^n\}$. This yields a set of link flows $\{x_a^n\}$.

Step 3 *Convergence test*. If $\max_{a \in A}\left\{\left|x_a^n - x_a^{n-1}\right| \le \kappa\right\}$, stop. Otherwise, let $n := n+1$ and go to Step 1.

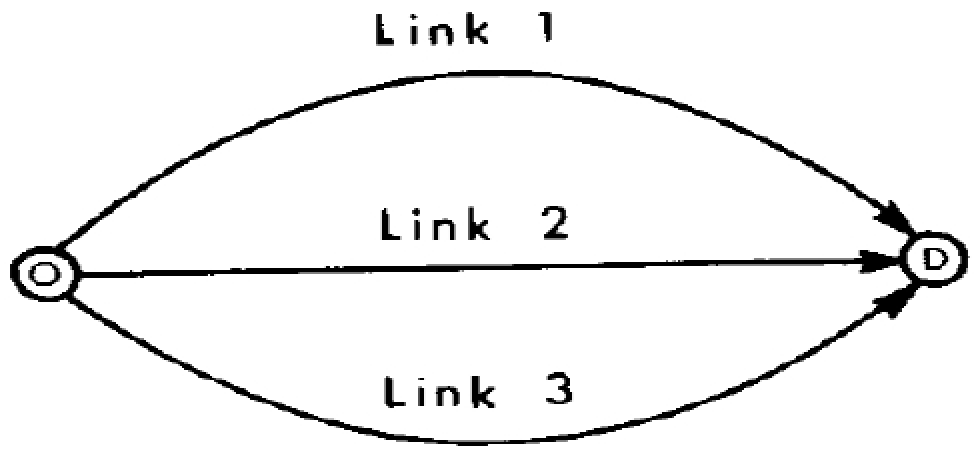


Figure 4.18b

$$t_1 = 10\left[1+0.15\left(\frac{x_1}{2}\right)^4\right] \quad \text{time units}$$

$$t_2 = 20\left[1+0.15\left(\frac{x_2}{4}\right)^4\right] \quad \text{time units}$$

$$t_3 = 25\left[1+0.15\left(\frac{x_3}{3}\right)^4\right] \quad \text{time units}$$

$$x_1 + x_2 + x_3 = 10 \quad \text{flow units}$$

Capacity Restraint Algorithm Applied to the Network in Figure 5.1

Iteration Number	Algorithmic Step	Link 1	Link 2	Link 3
0	Initialization	$t_1^0 = 10$	$t_2^0 = 20$	$t_3^0 = 25$
		$x_1^0 = 10$	$x_2^0 = 0$	$x_3^0 = 0$
1	Update	$t_1^1 = 947$	$t_2^1 = 20$	$t_3^1 = 25$
	Loading	$x_1^1 = 0$	$x_2^1 = 10$	$x_3^1 = 0$
2	Update	$t_1^2 = 10$	$t_2^2 = 137$	$t_3^2 = 25$
	Loading	$x_1^2 = 10$	$x_2^2 = 0$	$x_3^2 = 0$
3	Update	$t_1^3 = 947$	$t_2^3 = 20$	$t_3^3 = 25$
	Loading	$x_1^3 = 0$	$x_2^3 = 10$	$x_3^3 = 0$
		$\vdots$	$\vdots$	$\vdots$

- This algorithm may not converge;

- To remedy this converge problem, first, in stead of using the travel time obtained in the previous iteration for the new loading, a combination of the last two travel times obtained is used. This introduces a “smoothing” effect. Second, the iteration is made to stop after N iterations. The equilibrium flow pattern is taken to be the average flow for each link over the last four iterations. (N should never be less than 4.)

Step 0: Initialization. Perform all-or-nothing assignment based on $t_a^0 = t_a(0), \forall a$. Obtain a set of link flows $\{x_a\}$. Set iteration counter n=1

Step 1: Update. Set $\tau_a^n = t_a(x_a^{n-1}), \forall a$

Step 2: Smoothing. Set $t_a^n = 0.75 t_a^{n-1} + 0.25 \tau_a^n, \forall a$

Step 3: Network Loading. Assign all trips to the network using all-or-nothing based on travel times $\{t_a^n\}$. This yields a set of link flows $\{x_a^n\}$.

Step 4: Stopping rule. If $n = N$, go to stop 5. Otherwise, set n=n+1 and go to step 1.

Step 5: Averaging. Set $x_a^* = \frac{1}{4}\sum_{l=0}^{3} x_a^{n-l}, \forall a$ and stop. $\{x_a^*\}$ are the link flows at equilibrium.

- Note that this modified capacity restraint approach does not produce an equilibrium pattern.

Modified Restraint Algorithm Applied to the Network in Figure 5.1

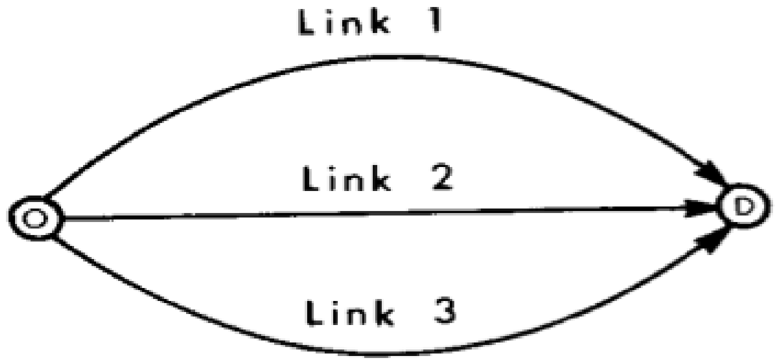
teration Number	Algorithmic Step	Link		
		1	2	3
0	Initialization	$t_1^0 = 10$	$t_2^0 = 20$	$t_3^0 = 25$
		$x_1^0 = 10$	$x_2^0 = 0$	$x_3^0 = 0$
1	Update	$\tau_1^1 = 947$	$\tau_2^1 = 20$	$\tau_3^1 = 25$
	Smoothing	$t_1^1 = 244$	$t_2^1 = 20$	$t_3^1 = 25$
	Loading	$x_1^1 = 0$	$x_1^1 = 10$	$x_3^1 = 0$
2	Update	$\tau_1^2 = 10$	$\tau_2^2 = 137$	$\tau_3^2 = 25$
	Smoothing	$t_2^2 = 186$	$t_2^2 = 49$	$t_3^2 = 25$
	Loading	$x_1^2 = 0$	$x_2^2 = 0$	$x_3^2 = 10$
3	Update	$\tau_1^3 = 10$	$\tau_2^3 = 20$	$\tau_3^3 = 488$
	Smoothing	$t_1^3 = 142$	$t_2^3 = 42$	$t_3^3 = 141$
	Loading	$x_1^3 = 0$	$x_2^3 = 10$	$x_3^3 = 0$
Average		$x_1^* = 2.5$	$x_2^* = 5.0$	$x_3^* = 2.5$
		$t_1^* = 13.7$	$t_2^* = 27.3$	$t_3^* = 26.8$

Incremental assignment

- Assigns a portion of the O-D entries at each iteration.
- Travel times are then updated and an additional portion of the O-D matrix is loaded onto the network.

```
Step 0: Preliminaries. Divide each origin-destination
        entry into N equal portions. (i.e. set
        $q_{rs}^n = q_{rs} / N$). Set n=1 and $x_a^0 = 0, \forall a$
Step 1: Update. Set $t_a^n = t_a(x_a^{n-1}), \forall a$
Step 2: Incremental loading. Perform all-or-nothing
        assignment based on $\{t_a^n\}$, but using only the
        trip rates $q_{rs}^n$ for each O-D pair. This yields a
        flow pattern $\{w_a^n\}$
Step 3: Flow summation. Set $x_a^n = x_a^{n-1} + w_a^n, \forall a$
Step 4: Stopping rule. If $n = N$, stop (the current set of
        link flows is the solution); otherwise, set
        n=n+1 and go to step 1.
```

Example



$$t_1 = 10\left[1+0.15\left(\frac{x_1}{2}\right)^4\right] \text{ time units}$$

$$t_2 = 20\left[1+0.15\left(\frac{x_2}{4}\right)^4\right] \text{ time units}$$

$$t_3 = 25\left[1+0.15\left(\frac{x_3}{3}\right)^4\right] \text{ time units}$$

$$x_1 + x_2 + x_3 = 10 \text{ flow units}$$

Incremental Assignment Algorithm (N=4)

Iteration	Algorithmic Step	output	Link		
			1	2	3
	updating	t	10	20	25
1	incremental loading	w	2.5	0	0
	summation	x	2.5	0	0
	updating	t	13.66211	20	25
2	incremental loading	w	2.5	0	0
	summation	x	5	0	0
	updating	t	68.59375	20	25
3	incremental loading	w	0	2.5	0
	summation	x	5	2.5	0
	updating	t	68.59375	20.45776	25
4	incremental loading	w	0	2.5	0
	summation	x	5	5	0
	travel time at convergence		68.59375	27.32422	25

- The order of loading the O-D pair is important.
- This is not an equilibrium assignment. With large number of iterations, the final flow pattern may approximate an UE pattern. But this is not guaranteed.

内容

用户均衡

$$\boxed{\begin{aligned}
&\left(c_{r,w}-\mu_w\right) f_{r,w}=0,\ \ r\in R_w, w\in W\\
&c_{r,w}-\mu_w\geq 0,\quad f_{r,w}\geq 0,\ \ r\in R_w, w\in W\\
&\sum_{r\in R_w} f_{r,w}=d_w, w\in W
\end{aligned}}$$

$$\Big\downarrow \qquad \begin{aligned}
&x_a=\sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ a\in A\\
&c_{r,w}=\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}, r\in R_w, w\in W
\end{aligned}$$

$$\boxed{\begin{aligned}
&\left(\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}-\mu_w\right) f_{r,w}=0,\ \ r\in R_w, w\in W\\
&\sum_{a\in A} t_a\left(x_a\right)\delta_{a,r}-\mu_w\geq 0,\ \ f_{r,w}\geq 0,\quad r\in R_w, w\in W\\
&\sum_{r\in R_w} f_{r,w}=d_w,\ \ w\in W\\
&x_a=\sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\quad a\in A
\end{aligned}}$$

贝克曼转化（Beckmann’s transformation）

If all link travel time functions t(x) are continuous, differentiable, increasing, separable，convex and additive, then the user equilibrium flow pattern is identical to the solution of the following optimization problem.

$$\left(\sum_{a\in A} t_a(x_a)\delta_{a,r} - \mu_w\right) f_{r,w} = 0,\ r\in R_w, w\in W$$

$$\sum_{a\in A} t_a(x_a)\delta_{a,r} - \mu_w \geq 0,\ \ f_{r,w}\geq 0,\ \ r\in R_w, w\in W$$

$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W$$

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ \ a\in A$$

$$\Longleftrightarrow$$

$$\min \sum_{a\in A}\int_0^{x_a} t_a(w)\,dw \qquad (1)$$

s.t.

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ \ a\in A \qquad (2)$$

$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W \qquad (3)$$

$$f_{r,w}\geq 0,\ \ r\in R_w, w\in W \qquad (4)$$

This equivalence can be demonstrated by proving that first-order conditions for the minimization program are identical to the equilibrium conditions.

贝克曼转化（Beckmann’s transformation）

Separable: $\frac{dt_a(x_a)}{dx_a} \geq 0, \quad \frac{dt_a(x_a)}{dx_b} = 0, \ a \neq b, a, b \in A$

Convex: $d^2 t_a(x_a) / dx_a{}^2 \geq 0, \ x_a \geq 0, \ a \in A$

Basic concept in minimization problem

Problem statement

Objective function: $\min z(x_1, \ldots, x_I)$

Subject to a set of constraints (which forms the feasible region)

$$g_1(x_1, \ldots, x_I) \geq b_1$$
$$g_2(x_1, \ldots, x_I) \geq b_2$$
$$\ldots$$
$$g_J(x_1, \ldots, x_I) \geq b_J$$

or can be written as:

$$\min z(\mathbf{x})$$

s.t.

$$g_j(\mathbf{x}) \geq b_j, \;\; j = 1, \ldots, J$$

The solution, $\mathbf{x}^*$, is a feasible value of $\mathbf{x}$ that minimizes $z(\mathbf{x})$

That is,

$$z(\mathbf{x}^*) \le z(\mathbf{x}) \text{ for any feasible } \mathbf{x}$$

$$g_j(\mathbf{x}^*) \ge b_j, \quad j = 1, \ldots, J$$

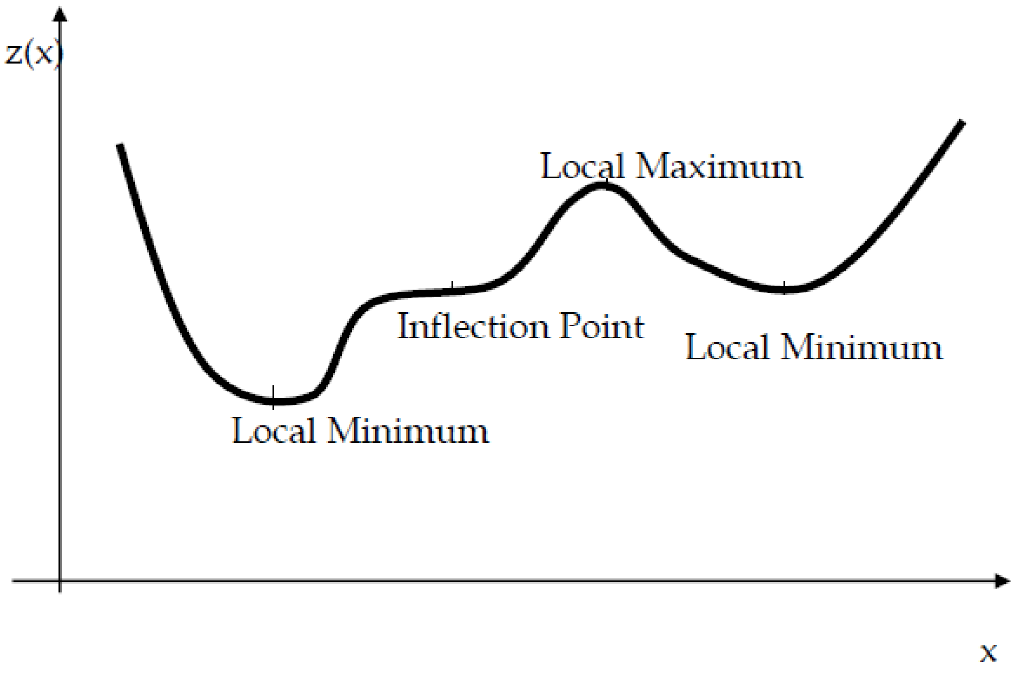
- "$\le$" constraints can be introduced by multiplying both sides by -1 to convert the constraints to "$\ge$" constraints.
- "=" constraints introduced by a pair of "$\le$" and "$\ge$" constraints.
- Maximization problems introduced by multiplying -1 to the objective function

Unconstrained minimization problem

Necessary Condition (or 1st order condition):

$$\frac{dz(x^*)}{dx} = 0,\ x^* \text{ is called a stationary point}$$

- The 1st order condition is not a sufficient condition. x^* can be a maximum, point of inflection.



Unconstrained minimization problem

- A sufficient condition for a stationary point to be a local minimum is for the function to be strictly convex in the vicinity of the stationary point.
- Strict convexity means that a line segment connecting any 2 points of the function lies entirely above the function

$$z[\theta x_1 + (1-\theta)x_2] < \theta z(x_1) + (1-\theta)z(x_2)$$

x_1, x_2 are any 2 points, and $0 < \theta < 1$

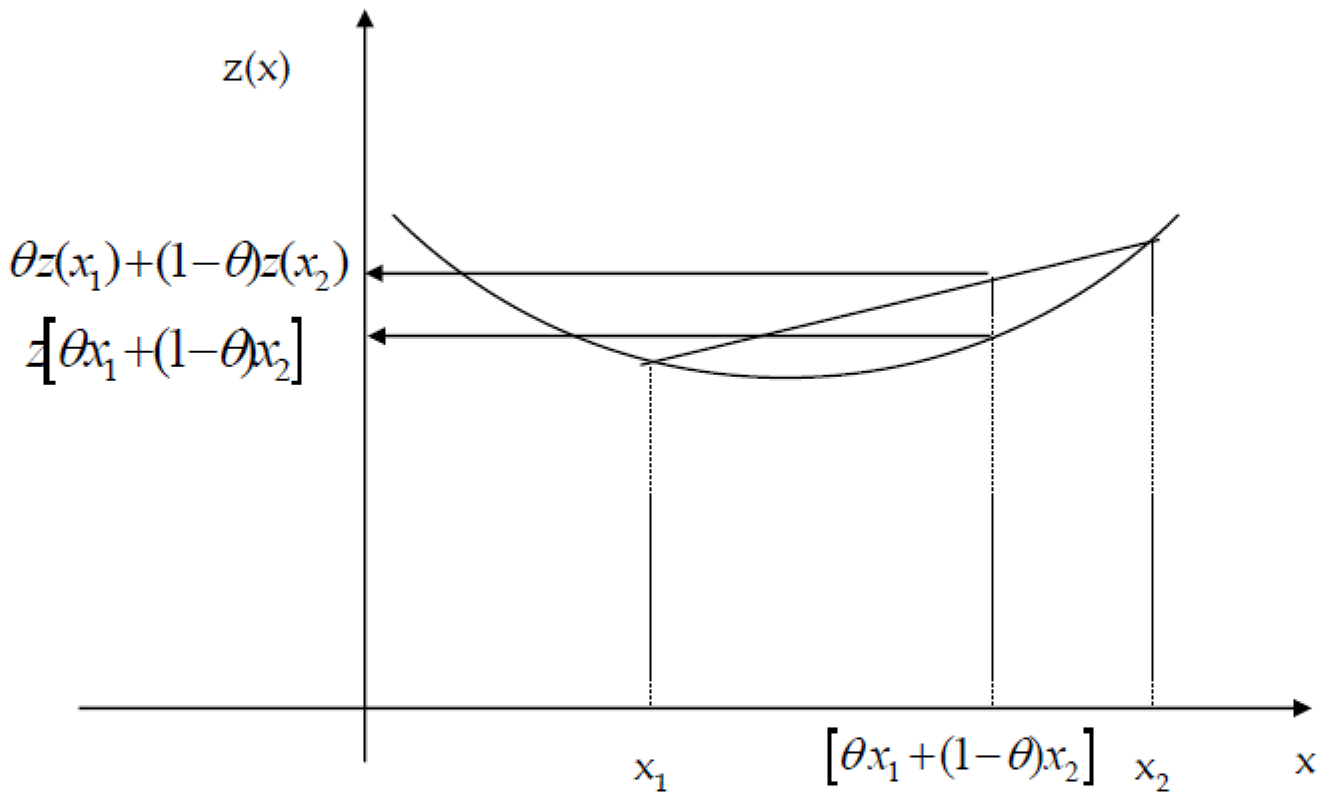
This condition does not require the function to be differentiable

- The if and only if condition for strict convexity can also be written as:

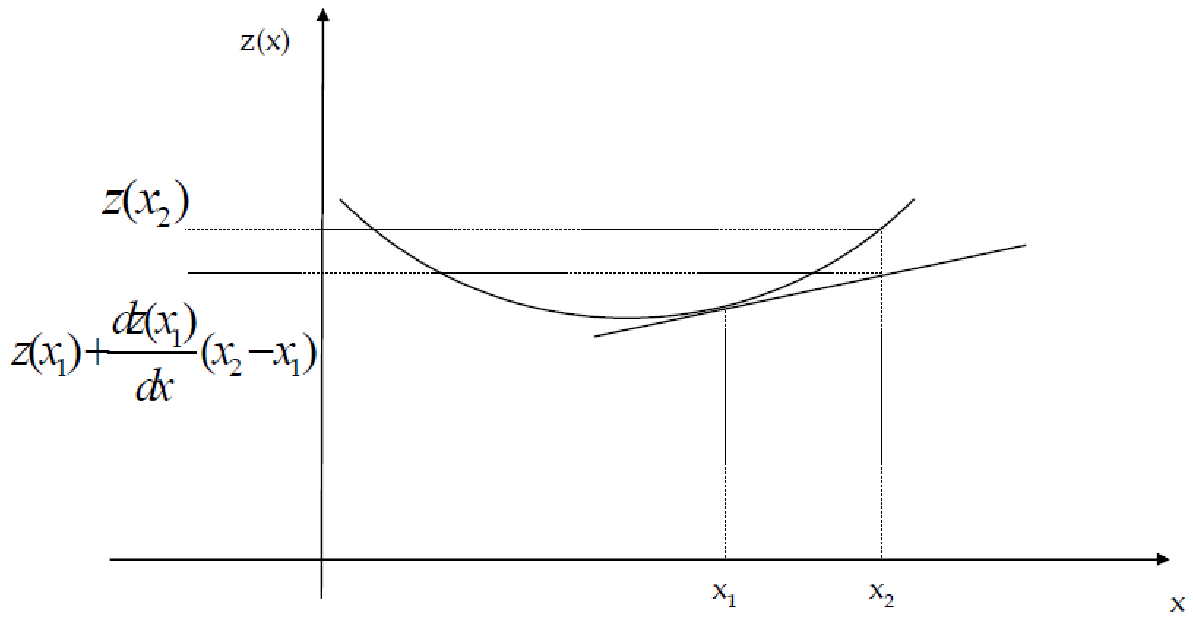
$$z(x_1) + \frac{dz(x_1)}{dx}(x_2 - x_1) < z(x_2)$$

or $$\frac{d^2 z(x^*)}{dx^2} > 0$$

Unconstrained minimization problem


z(x)
$\theta z(x_1)+(1-\theta)z(x_2)$
$z[\theta x_1+(1-\theta)x_2]$
x_1
$[\theta x_1+(1-\theta)x_2]$
x_2
x

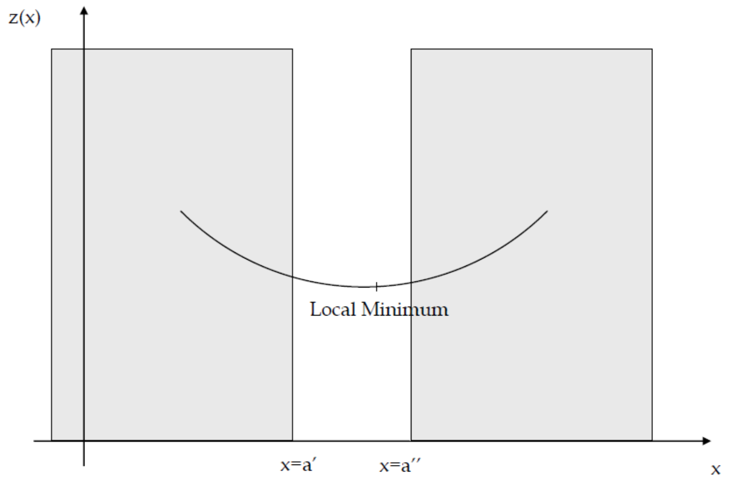
Unconstrained minimization problem



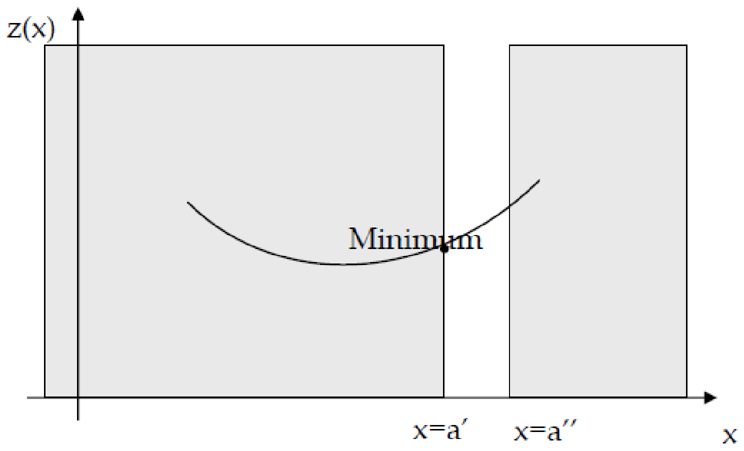
Unconstrained minimization problem

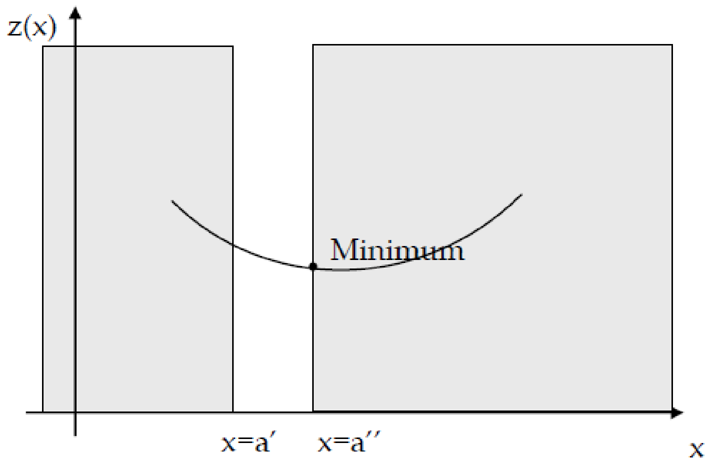
- If a function is strictly convex in the vicinity of the stationary point, this point is a local minimum
- To show that a point is a global minimum, one has to compare the value of all the local minima.
- Alternatively, one needs to show that x is unique or the only minimum by showing that z(x) is convex for all value of x (i.e., the function is globally convex.)

Constrained minimization problem



It is possible to have a minimum where the first derivative does not vanish.





Constrained minimization problem

- Note that for the last two cases, $\frac{dz(x^*)}{dx} \neq 0$, when the minimum occurs at the boundary of the binding constraint.
- If the binding constraint is of the type "$\geq$" then z(x) must be increasing or non-decreasing at x^*.
- If the binding constraint is of the type "$\leq$" then z(x) must be decreasing or non-increasing at x^*.

Constrained minimization problem

- Referring to figures (a), (b), and (c), and expressing the program in standard form:

$$\min z(x)$$
$$x \geq a'$$
$$-x \geq a''$$

The sign of $\frac{dz(x)}{dx}$ and derivation of the binding constraint at the minimum are the same:

Figure (b): $\boxed{\frac{d(x^*)}{dx}} = 1;\ \frac{dz(x^*)}{dx} > 0$

Figure (c): $\frac{d(-x^*)}{dx} = -1;\ \frac{dz(x^*)}{dx} < 0$

Constrained minimization problem

- Since both derivatives are scalars,

$$\frac{dz(x^*)}{dx} = u_j \frac{dg_j(x^*)}{dx}$$ where j is the $\boxed{\text{binding}}$ constraint

  u_j is a $\boxed{\text{nonnegative}}$ scalar; g_j is the binding constraint

- For the general situation, we can write:

$$\left[b_j - g_j(x^*)\right]u_j = 0 \quad j = 1, \ldots, J$$

  if u_j = 0, the constraint may or may not be binding

  if u_j $\geq$ 0, the constraint is binding

- Combining the same observations,

$$\frac{dz(x^*)}{dx} = \sum_j u_j \frac{dg_j(x^*)}{dx}$$

  $\underline{\text{Note that for an unconstrained program, } u_j = 0 \text{ for all j.}}$

Constrained minimization problem

- Summarizing, the first order conditions for a constrained minimum are:

$$\boxed{\begin{gathered}\frac{dz(x^*)}{dx} = \sum_j u_j \frac{dg_j(x^*)}{dx} \\ u_j \geq 0 \\ u_j\left[b_j - g_j(x^*)\right] = 0 \\ g_j(x^*) \geq b_j \quad j = 1, \ldots, J\end{gathered}}$$

Necessary condition

- Sufficient conditions require that
  1. Strict convexity in the vicinity of the local minimum
  2. Convexity of the objective function
  3. Convexity of the feasible region (a line segment connecting any 2 points of that feasible region must lie entirely within the region.

Multidimensional programs

Unconstrained Minimization Problems

- First order conditions: Necessary condition

Gradient $$\nabla_x z(\mathbf{x}) = \left(\frac{\partial z(\mathbf{x})}{\partial x_1}, \frac{\partial z(\mathbf{x})}{\partial x_2}; \dots, \frac{\partial z(\mathbf{x})}{\partial x_I} \right)$$

$$\nabla z(\mathbf{x}^*) = 0$$

or $$\frac{\partial z(\mathbf{x})}{\partial x_i} = 0, \ \forall i$$

Example:

$$z(x) = x_1^2 + 2x_2^2 + 2x_1x_2 - 2x_1 - 4x_2$$

- To show that the stationary point $\mathbf{x}^*$ is a local minimum, it is sufficient to show that $z(\mathbf{x}^*)$ is strictly convex. Sufficient condition

$$z[\theta\, \mathbf{x}_1 + (1-\theta)\mathbf{x}_2] < \theta\, z(\mathbf{x}_1) + (1-\theta) z(\mathbf{x}_2)$$

$\mathbf{x}_1, \mathbf{x}_2$ are any 2 points, and $0 < \theta < 1$

or the if and only if condition for strict convexity can be written as:

$$z(\mathbf{x}_1) + \nabla z(\mathbf{x}_1)(\mathbf{x}_2 - \mathbf{x}_1)^{\mathrm{T}} < z(\mathbf{x}_2)$$

Multidimensional programs

- A function $z(\mathbf{x})$ is strictly convex if the Hessian of $z(\mathbf{x})$

$$\nabla^2 z(\mathbf{x}) \equiv \begin{bmatrix} \frac{\partial^2 z(\mathbf{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 z(\mathbf{x})}{\partial x_1 \partial x_I} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 z(\mathbf{x})}{\partial x_I \partial x_1} & \cdots & \frac{\partial^2 z(\mathbf{x})}{\partial x_I^2} \end{bmatrix} \text{ is positive definite.}$$

- A matrix **H** is positive definite if

  $\mathbf{h} \cdot \mathbf{H} \cdot \mathbf{h}$ is positive for any nonzero vector **h**

- A symmetric matrix is positive definite if and only if its eigen values are positive.

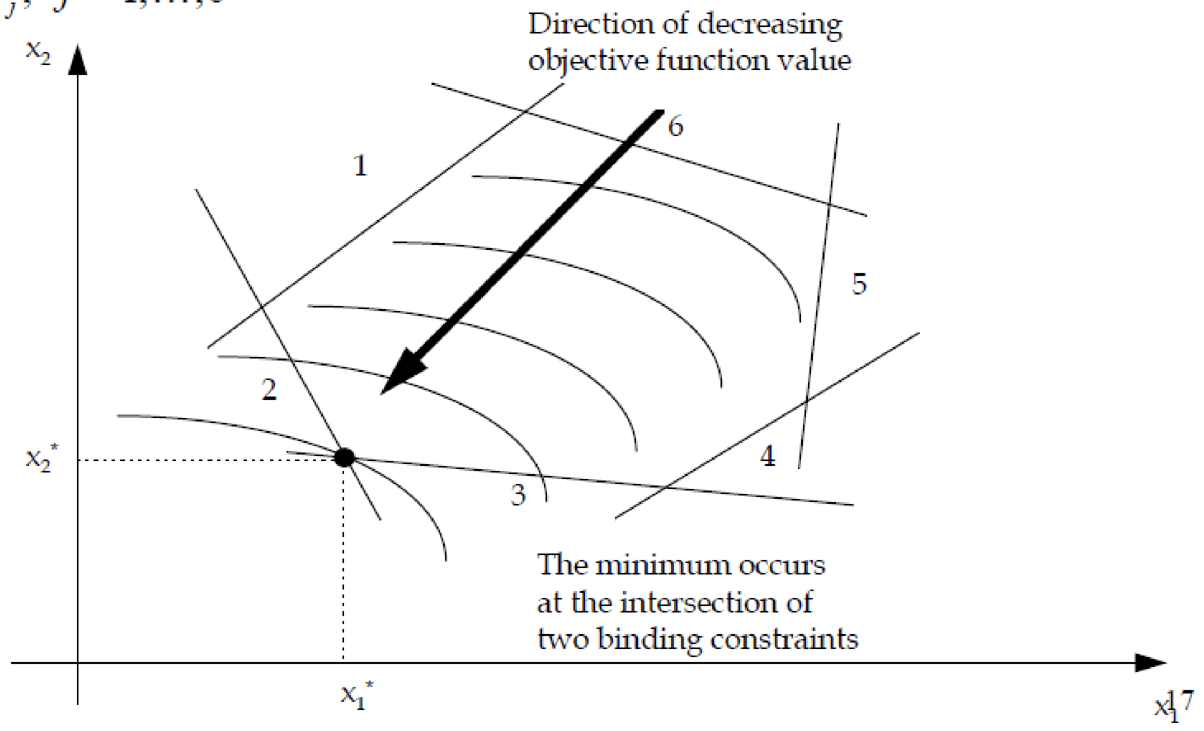
  (Eigenvalues are obtained by solving λ in $\mathbf{Ax} - \lambda\mathbf{x} = 0$)

- Since the matrix that we will deal with in this class is diagonal, positive definite in this case means that all the diagonal elements are positive. This implies that the objective function is strictly convex.

Multidimensional programs

Constrained minimization program

$$\min z(x_1, x_2)$$
$$g_j(x_1, x_2) \geq b_j, \ \ j = 1, \ldots, 6$$



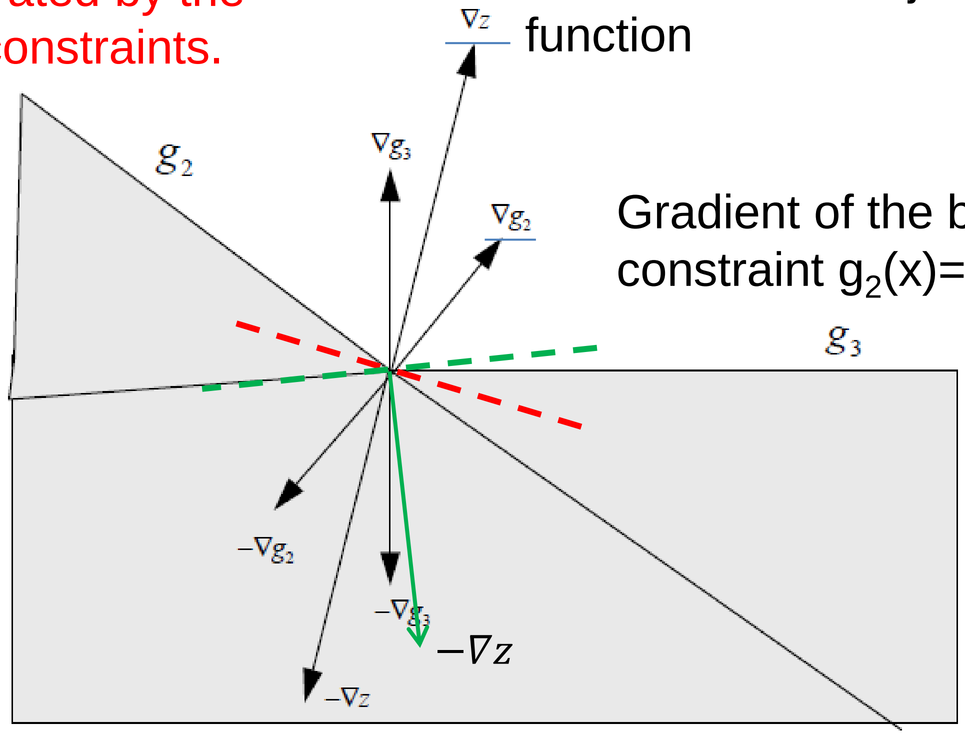
Direction of decreasing objective function value
x_2
x_2^*
x_1^*
x_1
1
2
3
4
5
6
The minimum occurs at the intersection of two binding constraints

Multidimensional programs

The gradient of the objective functions lies within the cone generated by the gradients of the binding constraints.

Gradient of the objective function

Gradient of the binding constraint $g_2(x)=b_2$



If $-\nabla z$ points below $-\nabla g_3$, then z(x) will decrease by sliding along g_3.

Similarly, if $-\nabla z$ points above $-\nabla g_2$ then z(x) will decrease by sliding along g_2. Therefore $-\nabla z$ must lie between $-\nabla g_2$ and $-\nabla g_3$.

Multidimensional programs

Mathematically, this can be written as:

$\nabla z(\mathbf{x}^*) = u_2 \nabla g_2(\mathbf{x}^*) + u_3 \nabla g_3(\mathbf{x}^*)$ where u_2, u_3 are nonnegative scalars.

- In general, the first order conditions can be written as:

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = \sum_j u_j \frac{\partial g_j(\mathbf{x}^*)}{\partial x_i} \quad i = 1, \ldots, I$$

$$u_j \geq 0$$

Dual variable $$u_j \left[b_j - g_j(\mathbf{x}^*) \right] = 0$$

$$g_j(\mathbf{x}^*) \geq b_j$$

where $u_j = 0$ for any non-binding constraints.

When the right hand side of the jth constraint is relaxed by Δb_j, the minimum value of z(x) will decrease by $u_j \Delta b_j$. So u_j is a measure of the sensitivity of the minimum value of z(x) to the restriction imposed by the jth constraints

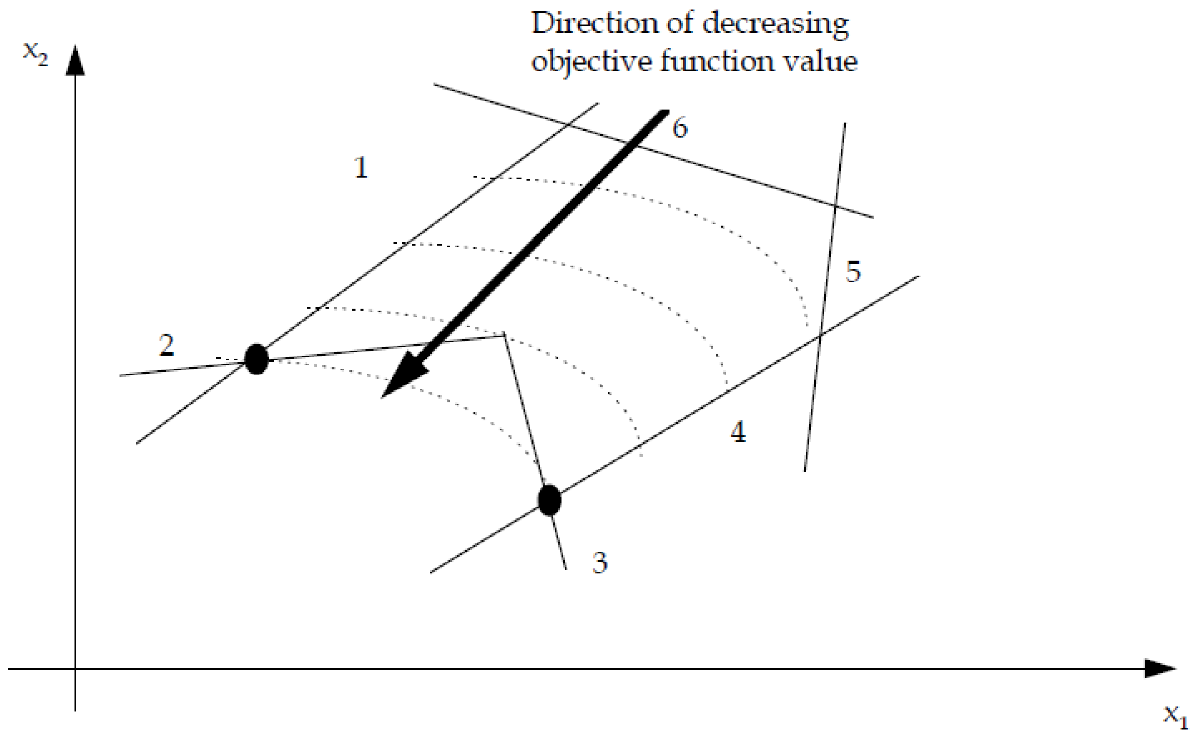
This is also called the Kuhn-Tucker Condition. u_j is called the dual variables or the Langrange multiplier. $u_j\left[b_j - g_j(\mathbf{x}^*)\right] = 0$ is called the complementary slackness condition.

Multidimensional programs

- The gradient of the objective function at $\mathbf{x}^*$ is a linear combination of the binding constraints
- If none constraint is binding, this becomes the unconstrained case
- Provided that the feasible region is convex, the Kuhn-Tucker condition defines the minimum of z(x).
- The value of the dual variables provide a measure of sensitivity of z(x) at its minimum to the constraints. Example, if $g_j(\mathbf{x}) \geq b_j$ is relaxed by a small amount, Δb_j, the minimum value of z(x) will decrease by $u_j(\Delta b_j)$. If a constraints is not binding, $u_j = 0$, therefore z(x) is not sensitive to that constraint.
- The second order conditions require that the feasible region is convex.

Multidimensional programs

An example of nonconvex feasible region. There may be multiple minima.



Summary

- If x* meets the first order conditions, then x* is a stationary point.
- If the objective function is strictly convex in the vicinity of a stationary point x*, then x* is the unique local minimum.
- If z(x) is convex for all feasible x, and the feasible region is also convex, then the locally optimal solution x* is a unique global optimal solution.

Some special programs

<u>Nonnegativity Constraints</u>

$$\boxed{\begin{array}{ll} & \min Z(x) \\ \text{s.t.} & x \geq 0 \end{array}}$$

The first conditions are:

$$x^* \frac{dz(x^*)}{dx} = 0 \text{ and } \frac{dz(x^*)}{dx} \geq 0$$

They imply either $x^* = 0$ (binding) or $\frac{dz(x^*)}{dx} = 0$ (non-binding)

Similarly, for multi-dimensional program, the first order conditions are:

$$x_i^* \frac{\partial z(\mathbf{x}^*)}{\partial x_i} = 0 \text{ and } \frac{\partial z(\mathbf{x}^*)}{\partial x_i} \geq 0$$

Some special programs

Linear Equity Constraints

$$\boxed{\begin{aligned} & \min Z(x) \\ \text{s.t.} \quad & \sum_i h_{ij} x_i = b_j \quad j = 1, \dots, J \end{aligned}} \qquad (*)$$

Example: $\min Z(x_1, x_2) = x_1 + x_2^2 + x_3$

s.t.

$$x_1 + 2x_2 = 3$$

$$x_1 + x_3 = 4$$

The Kuhn-Tucker conditions:

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = \sum_j u_j h_{ij} \quad i = 1, \dots, I$$

$$\sum_i h_{ij} x_i^* = b_j \quad j = 1, \dots, J$$

Some special programs

- At the solution, all constraints are binding. Therefore, the complementary slackness conditions are automatically satisfied. These first order conditions can also be derived by the method of Lagrange Multiplier.
- Construct the auxiliary function, the Lagrangian

$$L(x,u) = z(x) + \sum_j u_j \left[b_j - \sum_i h_{ij} x_i \right]$$

The stationary point of L(x,u) coincides with the constrained program of (*) above. Note that the Lagrangian is unconstrained. L(x,u) can be solved by

$$\nabla_x L(\mathbf{x}^*, \mathbf{u}^*) = 0$$

$$\nabla_u L(\mathbf{x}^*, \mathbf{u}^*) = 0$$

Some special programs

They imply

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = \sum_j u_j h_{ij} \quad i = 1, \ldots, I$$

$$\sum_i h_{ij} x_i^* = b_j \quad j = 1, \ldots, J$$

This is the same K-T conditions as before.

Note that at any feasible point, $\sum_i h_{ij} x_i^* = b_j \quad j = 1, \ldots, J$, therefore

L(x,u)=z(x).

Some special programs

<u>Nonnegativity and Linear Equality Constraints</u>

$$\min Z(x)$$

s.t. $$\sum_i h_{ij} x_i = b_j \quad j = 1, \ldots, J \qquad (*)$$

$$x_i \geq 0 \quad i = 1, \ldots, I$$

First we form the Lagrangian:

$$L(x,u) = z(x) + \sum_j u_j \left[b_j - \sum_i h_{ij} x_i \right]$$

s.t. $$x_i \geq 0 \quad i = 1, \ldots, I$$

Using earlier results, the first order conditions:

$$x_i^* \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_i} = 0 \text{ and } \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_i} \geq 0 \quad \forall i$$

$$\frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_j} = 0$$

Some special programs

The first order conditions can be written as:

$$x_i^* \left[\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j^* h_{ij} \right] = 0 \quad \forall i$$

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j^* h_{ij} \geq 0 \quad \forall i$$

$$\sum_i h_{ij} x_i^* = b_j \quad \forall j$$

$$x_i^* \geq 0 \quad \forall i$$

(We will use this set of conditions a lot in this course.)

贝克曼转化（**Beckmann’s transformation**）

If all link travel time functions t(x) are continuous, differentiable, increasing, convex, separable and additive, then the user equilibrium flow pattern is identical to the solution of the following optimization problem.

$$\left(\sum_{a\in A} t_a(x_a)\delta_{a,r} - \mu_w\right) f_{r,w} = 0,\ r\in R_w, w\in W$$
$$\sum_{a\in A} t_a(x_a)\delta_{a,r} - \mu_w \ge 0,\ \ f_{r,w}\ge 0,\ \ r\in R_w, w\in W$$
$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W$$
$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ \ a\in A$$

$$\Longleftrightarrow$$

$$\min \sum_{a\in A}\int_0^{x_a} t_a(w)\,dw \qquad (1)$$

s.t.

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ \ a\in A \qquad (2)$$
$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W \qquad (3)$$
$$f_{r,w}\ge 0,\ \ r\in R_w, w\in W \qquad (4)$$

This equivalence can be demonstrated by proving that first-order conditions for the minimization program are identical to the equilibrium conditions.

Preliminaries

Lagrangian (minimization problem with linear equality and non-negative constraints)

$$L(\mathbf{f},\mathbf{u}) = z\left[\mathbf{x}(\mathbf{f})\right] + \sum_{w\in W} \underbrace{\mu_w}_{\text{dual variable}} \left(d_w - \sum_{r\in R_w} f_{r,w} \right) \qquad (5)$$

s.t. $\mathbf{f} \geq \mathbf{0}$

$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W \qquad (3)$$

$$\min \sum_{a\in A} \int_0^{x_a} t_a(w)\,dw$$

s.t.

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\quad a\in A$$

$$\sum_{r\in R_w} f_{r,w} = d_w,\ \ w\in W$$

$$f_{r,w}\geq 0,\quad r\in R_w, w\in W$$

First-order conditions

$$f_{r,w}\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}}=0 \text{ and } \frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}}\geq 0 \tag{6}$$

For dual variables

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial \mu_w}=0, \forall w\in W \tag{7}$$

Non-negative constraints

$$f_{r,w}\geq 0, \forall r\in R_w, w\in W \tag{8}$$

Proof of equivalence

$$L(\mathbf{f},\mathbf{u}) = z\left[\mathbf{x}(\mathbf{f})\right] + \sum_{w\in W} \mu_w \left(d_w - \sum_{r\in R_w} f_{r,w} \right)$$

From (6) we obtain

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}} = \frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} + \frac{\partial}{\partial f_{r,w}} \sum_{w\in W} \mu_w \left(d_w - \sum_{r\in R_w} f_{r,w} \right)$$

$$\frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} = \qquad (9)$$

Proof of equivalence

From (6) we obtain

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}} = \frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} + \frac{\partial}{\partial f_{r,w}} \sum_{w\in W} \mu_w \left(d_w - \sum_{r\in R_w} f_{r,w} \right)$$

$$\frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} = \sum_{b\in A} \frac{\partial z(\mathbf{x})}{\partial x_b} \frac{\partial x_b}{\partial f_{r,w}} \xrightarrow{} \begin{cases} 1 \text{ if } r=m, s=n \text{ and } k=l; \\ 0 \text{ otherwise} \end{cases}$$

$$= \sum_{b\in A} \left(\frac{\partial}{\partial x_b} \left(\sum_{a\in A} \int_0^{x_a} t_a(\omega)\,\mathrm{d}\omega \right) \right) \delta_{b,r}$$

$$= \sum_{b\in A} t_b(x_b)\,\delta_{b,r} \xrightarrow{} \text{travel cost on that path}$$

$$= c_{r,w}$$

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}} = \frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} + \frac{\partial}{\partial f_{r,w}} \sum_{w\in W} \underbrace{\mu_w}_{\boxed{\text{not function of } f_{r,w}}} \left(\underbrace{d_w}_{\boxed{\text{constant}}} - \sum_{r\in R_w} f_{r,w} \right)$$

$$\frac{\partial}{\partial f_{r,w}} \sum_{w\in W} \mu_w \left(d_w - \sum_{r\in R_w} f_{r,w} \right) = -\mu_w \qquad (10)$$

Substitute (9) and (10) into (11)

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}} = \frac{\partial z(\mathbf{x}(\mathbf{f}))}{\partial f_{r,w}} + \frac{\partial}{\partial f_{r,w}} \sum_{w\in W} \mu_w \left(d_w - \sum_{r\in R_w} f_{r,w} \right) \quad (11)$$

$$\Big\Downarrow$$

$$\frac{\partial L(\mathbf{f},\mathbf{u})}{\partial f_{r,w}} = c_{r,w} - \mu_w \quad (12)$$

Recall (6) (7) and (8), the general first-order conditions for the minimization can be expressed explicitly as:

User Equilibrium conditions

$$f_{r,w}\left(c_{r,w}-\mu_w\right)=0, \forall r\in R_w, w\in W \quad (13)$$

If a path k is not used ($f_k^{rs}=0$), then $c_k^{rs}-u_{rs}\geq 0$

On the other hand, if $f_k^{rs}>0$, then $c_k^{rs}-u_{rs}=0$

$$c_{r,w}-\mu_w\geq 0, \forall r\in R_w, w\in W \quad (14)$$

u_{rs} is the minimum travel cost between O-D pair *r*-*s*

$$\sum_{r\in R_w} f_{r,w}=d_w,\ w\in W \quad (15)$$

flow conservation

$$f_{r,w}\geq 0,\quad r\in R_w, w\in W \quad (16)$$

non-negative

Uniqueness Conditions

- The feasible region is convex.
- Objective function is convex (Hessian matrix of function z)

$$\frac{\partial^2 z(\mathbf{x})}{\partial x_m \partial x_n} = \frac{\partial t_m(x_m)}{\partial x_n} = \begin{cases} \frac{\partial t_n(x_n)}{\partial x_n} & \text{for } m = n \\ 0 & \text{otherwise} \end{cases} \Rightarrow \nabla^2 z(\mathbf{x}) = \begin{bmatrix} \frac{dt_1(x_1)}{dx_1} > 0 & 0 & 0 & \cdots \\ 0 & \frac{dt_2(x_2)}{dx_2} > 0 & 0 & \cdots \\ 0 & 0 & \ddots & \cdots \\ \vdots & \vdots & \vdots & 0 < \frac{dt_{|A|}(x_{|A|})}{dx_{|A|}} \end{bmatrix}$$

Uniqueness Conditions

If the travel time function $t_a(x_a)$ on any link $a \in A$ is a strictly increasing function of link flow x_a, a link flow pattern **x*** that satisfies the KKT conditions of Beckmann's transformation is the unique UE link flow solution. However, the optimal path flow pattern may not be unique.

Any path flow pattern satisfying the following conditions is optimal path flow pattern:

$$x_a^* = \sum_{w \in W} \sum_{r \in R_w} f_{r,w} \delta_{a,r}, \quad a \in A$$

$$\sum_{r \in R_w} f_{r,w} = d_w, \ \ w \in W$$

$$f_{r,w} \geq 0, \quad r \in R_w, w \in W$$

Descent algorithms for minimization problem

Basic concept

- The core of any algorithmic procedure is the calculation of $\boldsymbol{x}^{n+1}$ from $\boldsymbol{x}^{n}$:

$$\boldsymbol{x}^{n+1} = \boldsymbol{x}^{n} + \alpha_n \boldsymbol{d}^{n}$$

  where $\boldsymbol{d}^{n}$ is a descent direction vector, and α_n is a nonnegative scalar known as move size (or step size).

- Any descent direction can be characterized by the inequality:

$$\nabla \boldsymbol{z} \cdot \boldsymbol{d} < 0$$

- The move size is typically chosen so that the objective function is minimized along the descent direction.

Descent direction

$\nabla z \cdot \boldsymbol{d} < 0$

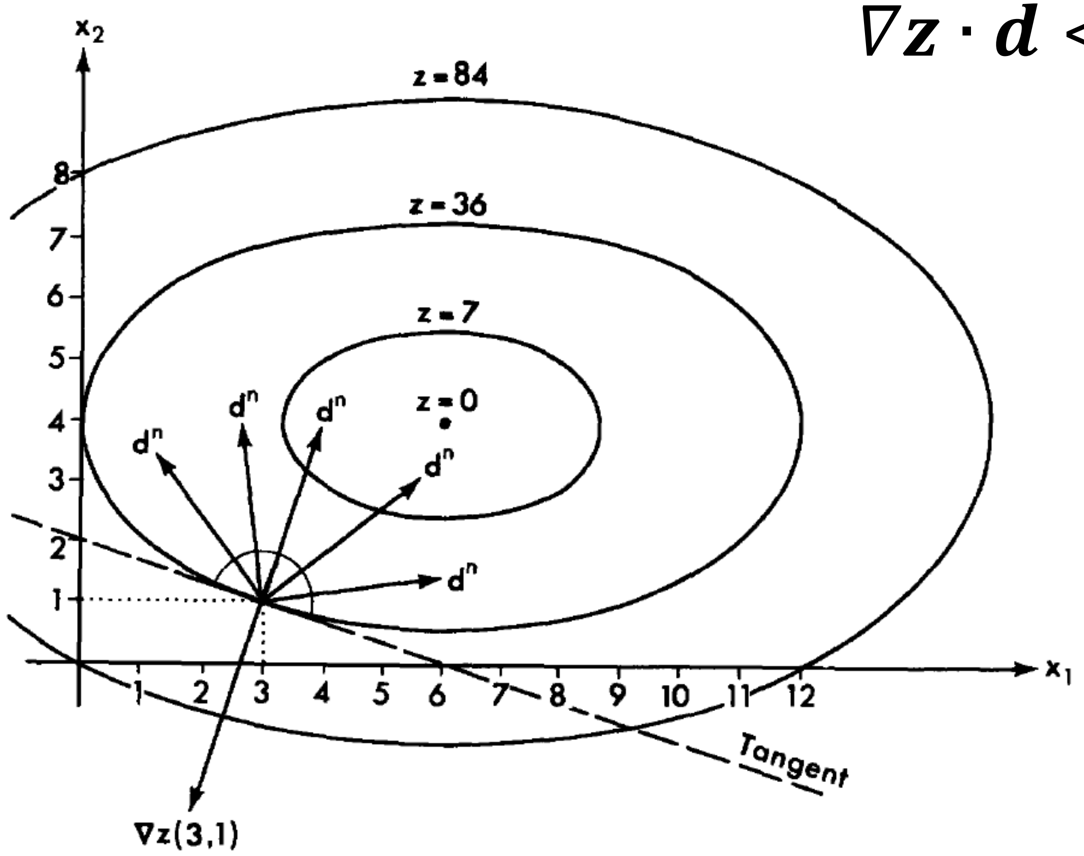


Figure 4.4 Contour lines of $z(x_1, x_2) = (x_1 - 6)^2 + 3(x_2 - 4)^2$ and the range of descent directions at $(x_1, x_2) = (3, 1)$.

Convex combination method

- The convex combination method was originally suggested by Frank and Wolfe in 1956 as a procedure for solving quadratic programming problems with linear constraints, and is known also as the Frank-Wolfe (FW) method.
- It is a feasible direction method. It starts from an initial feasible solution, and maintains the feasibility of solutions after each iteration.

Convex combination method

The main step of the Frank-Wolfe algorithm is formally stated as follows. At iteration k,

1. Find a descent direction $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$. $\mathbf{y}^k$ is the optimal solution to the linear program

$$\min \bar{z}(\mathbf{y}) = z(\cancel{\mathbf{x}^k}) + \langle \nabla z(\mathbf{x}^k), \mathrm{y} - \cancel{\mathbf{x}^k} \rangle$$
$$\text{subject to: constraints in the original problem}$$

2. Find an optimal step size. Find α^* that solves the one-dimensional nonlinear program

$$\min z(\mathbf{x}^k + \alpha(\mathbf{y}^k - \mathbf{x}^k))$$
$$\text{subject to: } 0 \leq \alpha \leq 1$$

3. Move. Set $x^{k+1} = x^k + \alpha^*(y^k - x^k)$

Convex combination method

- The convex combinations algorithm involves a minimization of a linear program as part of the direction-finding step, therefore is useful only in cases in which this linear program can be solved relatively easily.

Example

Let's try to use the Frank-Wolfe algorithm to solve the following problem.

$$\min z(\mathbf{x}) = x_1^2 + 2x_2^2 - 2x_1x_2 - 10x_2$$

subjec to

$$0 \le x_1 \le 4$$
$$0 \le x_2 \le 6$$

At a given solution $\mathbf{x}^k$, the linearized sub-problem reads

$$\min z^k(\mathbf{y}) = (2x_1^k - 2x_2^k)y_1 + (4x_2^k - 2x_1^k - 10)y_2$$

subjec to

$$0 \le y_1 \le 4$$
$$0 \le y_2 \le 6$$

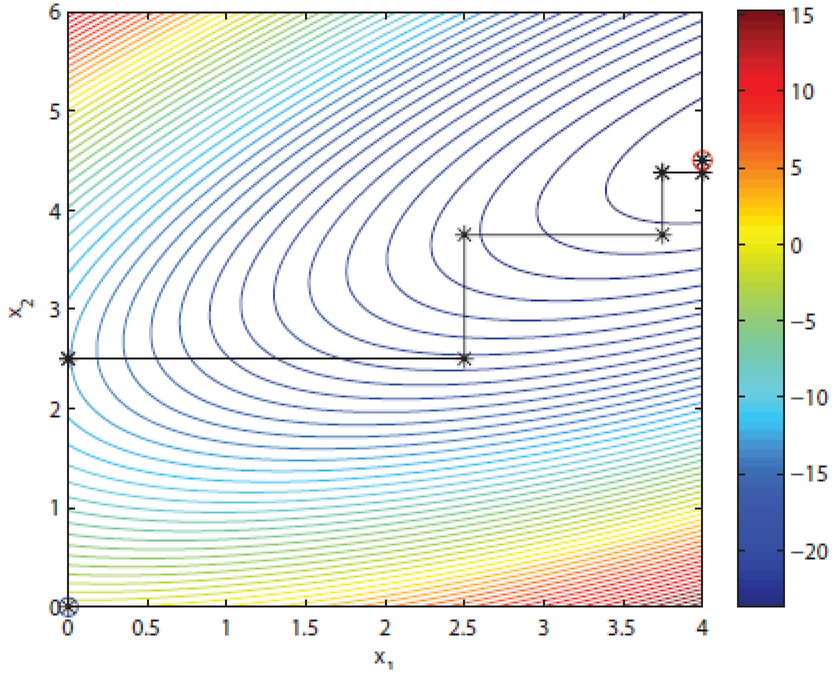
Once $\mathbf{y}^k$ is obtained, we set the descent direction $\mathbf{d}^k = \mathbf{y}^k - \mathbf{x}^k$. The new solution can be written as $\mathbf{x}^\alpha = \mathbf{x}^k + \alpha\mathbf{d}^k$. Then we attempt to find α such that the following function is minimized.

$$z(\alpha) = (x_1^\alpha)^2 + 2(x_2^\alpha)^2 - 2x_1^\alpha x_2^\alpha - 10x_2^\alpha$$

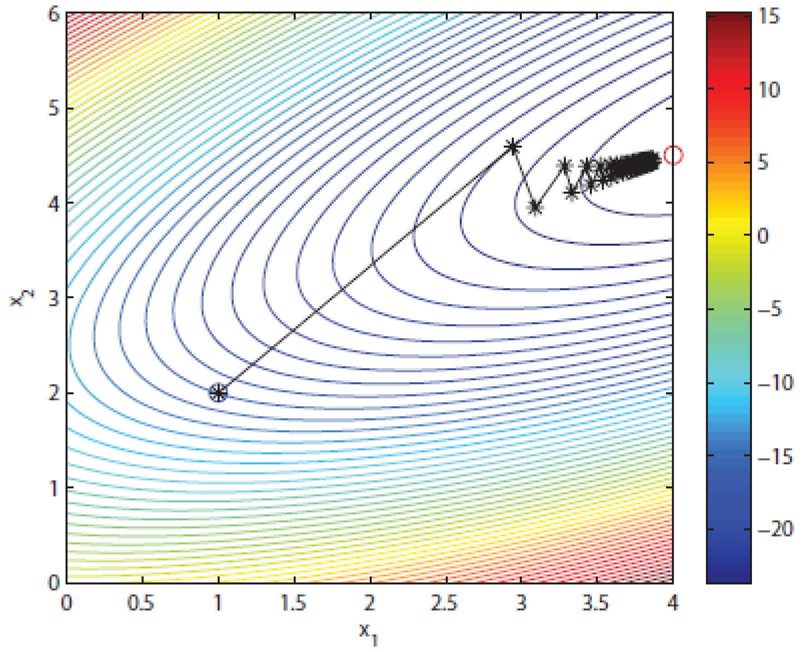
Since this is a quadratic function, we can simply set the following to zero

$$\frac{dz(\alpha)}{d\alpha} = (2x_1^\alpha - 2x_2^\alpha)(y_1^k - x_1^k) + (4x_2^\alpha - 2x_1^\alpha - 10)(y_2^k - x_2^k) = 0$$

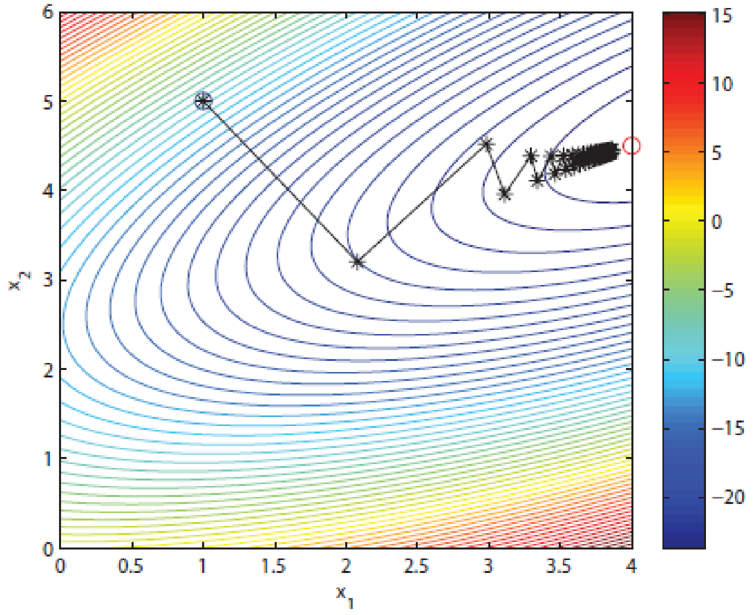
- When $x^0 = [0, 0]$, the algorithm converges in eight iterations, with an objective function value of -24.5 and a gap of 0. The algorithm finds the "true" solution after 8 iterations.



- When $x^0 = [0, 0]$, the algorithm converges in eight iterations, with an objective function value of -24.5 and a gap of 0. The algorithm finds the "true" solution after 8 iterations.
- When $x^0 = [1, 2]$, the algorithm terminates after 50 iterations, with an objective function value of -24.355244, and a gap = 0.631553. In this case, the algorithm enters a slow convergence mode once it gets close to the optimal solution.



- When $x^0 = [0, 0]$, the algorithm converges in eight iterations, with an objective function value of -24.5 and a gap of 0. The algorithm finds the "true" solution after 8 iterations.
- When $x^0 = [1, 2]$, the algorithm terminates after 50 iterations, with an objective function value of -24.355244, and a gap = 0.631553. In this case, the algorithm enters a slow convergence mode once it gets close to the optimal solution.
- When $x^0 = [1, 5]$, the algorithm terminates after 50 iterations, with an objective function value of -24.353079, and a gap = 0.212373. Again, the algorithm becomes much slower in later stage.



The slow-converging final phase is a signature of the algorithm, known as the zigzagging effect.

Solving the UE problem with F-W algorithm

- The F-W algorithm was published in Naval Research Logistics Quarterly in 1956 when Philip Wolfe was teaching at Princeton.
- In the same year, Martin Beckmann, along with two co-authors, published Studies in the Economics of Transportation in which they wrote down the first TAP formulation.
- For more than a decade, no algorithms were known to be able to solve the large-scale UE-TAP effectively.
- The breakthrough came in early 70s', when researchers (notably Larry Leblanc from Northwestern and S. Nguen/M. Florian from Montreal) began to realize the F-W algorithm is a perfect solver for large TAP.
- The F-W algorithm has since become the "standard" TAP solver and has been implemented in almost all commercial software systems.

Solving the UE problem with F-W algorithm

Recall the Beckmann's transformation:

$$\min Z(\mathbf{x}) = \sum_{a \in A} \int_0^{x_a} t_a(w)\, dw$$

s.t.

$$x_a = \sum_{w \in W} \sum_{r \in R_w} f_{r,w} \delta_{a,r}, \quad a \in A$$

$$\sum_{r \in R_w} f_{r,w} = d_w, \ \ w \in W$$

$$f_{r,w} \geq 0, \quad r \in R_w, w \in W$$

Solving the UE problem with F-W algorithm

- According to the FW algorithm, a descent direction is given by $d^k=y^k-x^k$, where y^k solves the following linear programming:

$$\min z^n(\mathbf{y}) = \nabla z(\mathbf{x}^n)\cdot \mathbf{y}^T = \sum_a \frac{\partial z(\mathbf{x}^n)}{\partial x_a} y_a$$

s.t.

$$y_a = \sum_{w\in W}\sum_{r\in R_w} g_{r,w}\delta_{a,r}, \quad a\in A$$

$$\sum_{r\in R_w} g_{r,w} = d_w, \quad w\in W$$

$$g_{r,w}\geq 0, \quad r\in R_w, w\in W$$

$$\xrightarrow{\frac{\partial Z(\mathbf{x}^n)}{\partial x_a} = t_a\left(x_a^n\right) = t_a^n}$$

$$\min z^n(\mathbf{y}) = \sum_a t_a^n y_a$$

s.t.

$$y_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r}, \quad a\in A$$

$$\sum_{r\in R_w} f_{r,w} = d_w, \quad w\in W$$

$$f_{r,w}\geq 0, \quad r\in R_w, w\in W$$

A multi-commodity minimum cost flow problem with no link capacity constraint

Solving the UE problem with F-W algorithm

- According to the FW algorithm, a descent direction is given by $d^k=y^k-x^k$, where y^k solves the following linear programming:

$$\min z^n(\mathbf{y}) = \nabla z(\mathbf{x}^n)\cdot \mathbf{y}^T = \sum_a \frac{\partial z(\mathbf{x}^n)}{\partial x_a} y_a$$

s.t.

$$y_a = \sum_{w\in W}\sum_{r\in R_w} g_{r,w}\delta_{a,r}, \quad a\in A$$

$$\sum_{r\in R_w} g_{r,w} = d_w, \ \ w\in W$$

$$g_{r,w}\geq 0, \quad r\in R_w, w\in W$$

$$\frac{\partial Z(\mathbf{x}^n)}{\partial x_a} = t_a(x_a^n) = t_a^n$$

$$\min z^n(\mathbf{g}) = \sum_{w\in W}\sum_{r\in R_w}\left(\sum_{a\in A} t_a^n \delta_{a,r}\right) g_{r,w}$$

$$= \sum_{w\in W}\sum_{r\in R_w} c_{r,w}^n g_{r,w}$$

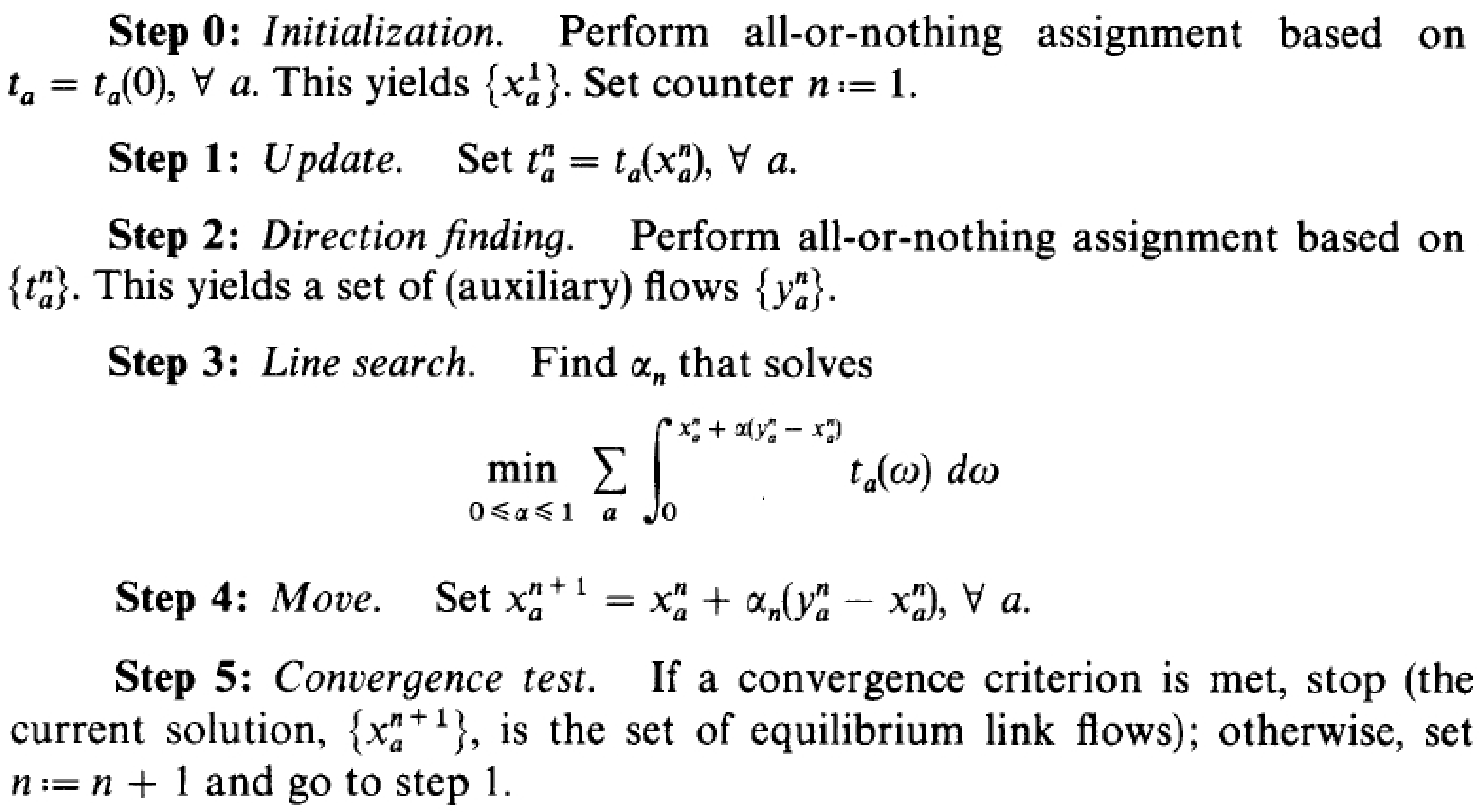
s.t.

$$\sum_{r\in R_w} g_{r,w} = d_w, \ \ w\in W$$

$$g_{r,w}\geq 0, \quad r\in R_w, w\in W$$

Therefore, by performing the following all-or-nothing traffic assignment algorithm we can obtain the auxiliary flow y^n. And $d^n=y^n-x^n$ is then a descent direction.

Solving the UE problem with F-W algorithm



Step 0: *Initialization.* Perform all-or-nothing assignment based on $t_a = t_a(0)$, $\forall$ a. This yields $\{x_a^1\}$. Set counter $n := 1$.

Step 1: *Update.* Set $t_a^n = t_a(x_a^n)$, $\forall$ a.

Step 2: *Direction finding.* Perform all-or-nothing assignment based on $\{t_a^n\}$. This yields a set of (auxiliary) flows $\{y_a^n\}$.

Step 3: *Line search.* Find α_n that solves

$$\min_{0 \leqslant \alpha \leqslant 1} \sum_a \int_0^{x_a^n + \alpha(y_a^n - x_a^n)} t_a(\omega)\, d\omega$$

Step 4: *Move.* Set $x_a^{n+1} = x_a^n + \alpha_n(y_a^n - x_a^n)$, $\forall$ a.

Step 5: *Convergence test.* If a convergence criterion is met, stop (the current solution, $\{x_a^{n+1}\}$, is the set of equilibrium link flows); otherwise, set $n := n + 1$ and go to step 1.

If the move size, α_n, is fixed at $\alpha_n = 1$ for all *n,* the resulting algorithm is identical to the capacity restraint method. This observation is important because it means that existing computer programs that use the capacity restraint method can be easily modified to give a convergent algorithmic solution to the UE problem (by inserting step 3).

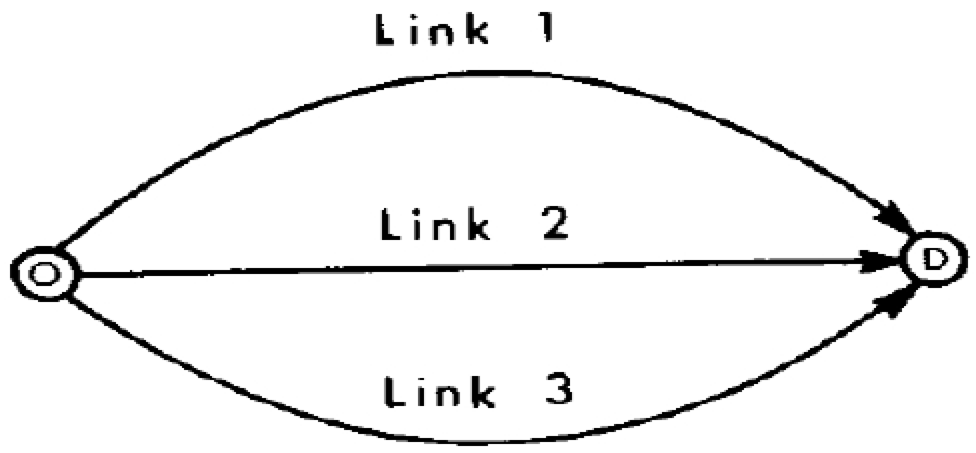


Figure 4.18b

$$t_1 = 10\left[1+0.15\left(\frac{x_1}{2}\right)^4\right] \quad \text{time units}$$

$$t_2 = 20\left[1+0.15\left(\frac{x_2}{4}\right)^4\right] \quad \text{time units}$$

$$t_3 = 25\left[1+0.15\left(\frac{x_3}{3}\right)^4\right] \quad \text{time units}$$

$$x_1 + x_2 + x_3 = 10 \quad \text{flow units}$$

TABLE 5.4 Convex Combinations Algorithm Applied to the Network in Figure 5.1

Iteration Number	Algorithmic Step	Link 1	Link 2	Link 3	Objective Function	Step Size
0	Initialization	$t_1^0 = 10.0$	$t_2^0 = 20.0$	$t_3^0 = 25.0$		
		$x_1^1 = 10.00$	$x_2^1 = 0.00$	$x_3^1 = 0.00$		
1	Update	$t_1^1 = 947.0$	$t_2^1 = 20.0$	$t_3^1 = 25.0$	$z(\mathbf{x}) = 1975.00$	
	Direction	$y_1^1 = 0$	$y_2^1 = 10$	$y_3^1 = 0$		$\alpha_1 = 0.596$
	Move	$x_1^2 = 4.04$	$x_2^2 = 5.96$	$x_3^2 = 0.00$		
2	Update	$t_1^2 = 35.0$	$t_2^2 = 35.0$	$t_3^2 = 25.0$	$z(\mathbf{x}) = 197.00$	
	Direction	$y_1^2 = 10$	$y_2^2 = 0$	$y_3^2 = 0$		$\alpha_1 = 0.161$
	Move	$x_1^3 = 3.39$	$x_2^3 = 5.00$	$x_3^3 = 1.61$		
3	Update	$t_1^3 = 22.3$	$t_2^3 = 27.3$	$t_3^3 = 35.3$	$z(\mathbf{x}) = 189.98$	
	Direction	$y_1^3 = 10$	$y_2^3 = 0$	$y_3^3 = 0$		$\alpha = 0.035$
	Move	$x_1^4 = 3.62$	$x_2^4 = 4.83$	$x_3^4 = 1.55$		
4	Update	$t_1^4 = 26.1$	$t_2^4 = 26.3$	$t_3^4 = 25.3$	$z(\mathbf{x}) = 189.44$	
	Direction	$y_1^4 = 0$	$y_2^4 = 0$	$y_3^4 = 10$		$\alpha = 0.020$
	Move	$x_1^5 = 3.54$	$x_2^5 = 4.73$	$x_3^5 = 1.72$		
5	Update	$t_1^5 = 24.8$	$t_2^5 = 25.8$	$t_3^5 = 25.4$	$z(\mathbf{x}) = 189.33$	
	Direction	$y_1^5 = 10$	$y_2^5 = 0$	$y_3^5 = 0$		$\alpha = 0.007$
	Move	$x_1^6 = 3.59$	$x_2^6 = 4.70$	$x_3^6 = 1.71$		
	Update	$t_1^6 = 25.6$	$t_2^6 = 25.7$	$t_3^6 = 25.4$	$z(\mathbf{x}) = 189.33$	

System optimal (SO)

- The system optimum assignment is based on **Wardrop's second principle:**

  Drivers cooperate with one another in order to minimize total system travel time.

- Obviously, this is not a behaviorally realistic model, but it can be useful to transport planners and engineers, trying to manage the traffic to minimize travel costs and therefore achieve an optimum social equilibrium.

System optimal (SO)

- The SO flow pattern solves the following optimization problem:

$$\min \sum_{a\in A} t_a\left(x_a\right)x_a$$

s.t.

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r}, \quad a\in A$$

$$\sum_{r\in R_w} f_{r,w} = d_w, \quad w\in W$$

$$f_{r,w}\geq 0, \quad r\in R_w, w\in W$$

UE flow pattern solves:

$$\min \sum_{a\in A}\int_0^{x_a} t_a\left(w\right)dw$$

s.t.

$$x_a = \sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r}, \quad a\in A$$

$$\sum_{r\in R_w} f_{r,w} = d_w, \quad w\in W$$

$$f_{r,w}\geq 0, \quad r\in R_w, w\in W$$

System optimal assignment

Example

Let us suppose a case where travel time is not a function of flow as shown in other words it is constant as shown in the figure below.

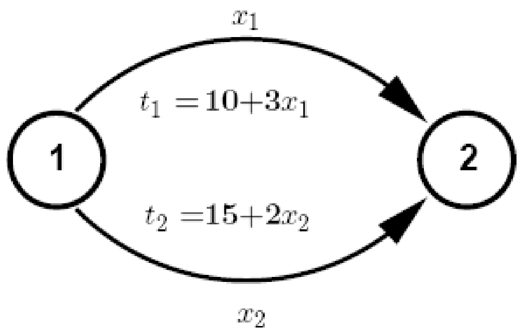


Figure 10:3: Two Link Problem with constant travel time function

$$\min (10 + 3x_1) x_1 + (15 + 2x_2) x_2$$

$$\text{s.t. } x_1 + x_2 = 12$$

SO v.s. UE

- The SO and UE flow patterns are generally not the same, unless the link travel time function is linear.
- The difference between SO and UE flow patterns validates the necessity of traffic regulation.

How to calculate the SO flow pattern?

- The KKT conditions of the SO problem:

$$\left(\sum_{a\in A}\left[t_a\left(x_a\right)+x_a t_a'\left(x_a\right)\right]\delta_{a,r}-\mu_w\right)f_{r,w}=0,\ \ r\in R_w, w\in W$$

$$\sum_{a\in A}\left[t_a\left(x_a\right)+x_a t_a'\left(x_a\right)\right]\delta_{a,r}-\mu_w\geq 0,\ \ f_{r,w}\geq 0,\ \ r\in R_w, w\in W$$

$$\sum_{r\in R_w} f_{r,w}=d_w,\ \ w\in W$$

$$x_a=\sum_{w\in W}\sum_{r\in R_w} f_{r,w}\delta_{a,r},\ \ a\in A$$

- Comparing the above KKT conditions with the UE conditions, we can see that the SO condition can be perceived as the UE conditions with the link travel time function of each link given by: $\tilde{t}_a\left(x_a\right)=t_a\left(x_a\right)+x_a t_a'\left(x_a\right)$